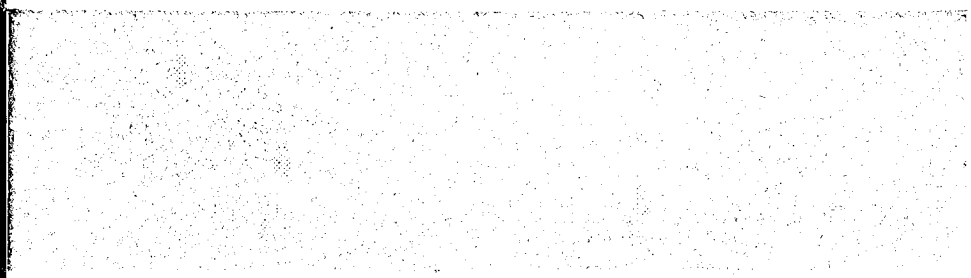

NASA

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

ARC 275 (Rev Feb 81)

#63

Panel Library Programmer's Manual

David A. Tristram[1], Pamela P. Walatka[2] and Eric L. Raible[3]

Report RNR-90-006, April 1990

NAS Systems Division
NASA Ames Research Center
Mail Stop 258-6
Moffett Field, CA 94035-1000

Preliminary Draft
Reviewer's Copy

---

[1] Silicon Graphics International, 2021 N Shoreline Bl., Mountain View, CA 94039-7311
[2]Computer Sciences Corporation, NASA Contract NAS 2-12961, Moffett Field, CA 94035-1000
[3]NASA Ames Research Center, Moffett Field, CA 94035-1000

README

The Panel Library is supported on all models of IRIS, and is currently most useful with C code. The manual and software are distributed without copyright to any institution or business within the United States. The software is distributed for the use of the recipient only and is not to be redistributed.

GETTING THE PANEL LIBRARY

All e-mail correspondences about the Panel Library should be addressed to

panel-request@nas.nasa.gov

We will e-mail instructions on how to get the code.
Or send regular mail to:

NAS Applied Research Office
ATTN: PANEL LIBRARY REQUEST
M/S T045-1
NASA Ames Research Center
Moffett Field, CA 94035

DOCUMENTATION

Hard copies of this manual are available from

NAS Documentation Center
M/S 258-6
NASA Ames Research Center
Moffett Field, CA 94035-1000
(415) 604-4632
(FTS) 464-4632
doc-center@prandtl.nas.nasa.gov

You can print your own from the file **manual.dvi** in the Panel Library directory. To print it, type: lpr -d -P*printername* manual.dvi. For example:

```
lpr -d -Pim5 manual.dvi
```

We strongly recommend that you pick up an already-printed copy from the Documentation Center.
Theoretically, the file **manual.dvi** is on-line viewable with the utility dviiris. Refer to Appendix A. [Please let us know if this works.]

PANEL EDITOR - EASY PANEL MAKING!

While the Panel Library has been designed to make it easy to create interactive panels, it is nevertheless time-consuming to make a panel look exactly the way you want. Eric Raible is working on a tool to solve this problem, a panel-library based graphical panel editor. With it, you can create panels and actuators dynamically, move them around, resize them, etc, and then dump out a ready-to-compile C program. Among many other features, it has extensive context sensitive on-line help. While it cannot at present edit existing panel library applications, this capability is planned for the future. You can get more information about the panel editor by sending a message to panel-request@nas.nasa.gov.

TUTORIAL

Karen McCann has prepared a Panel Library tutorial which is very helpful to Panel Library programmers. Contact Tom Kropp at Sterling Software (415) 964-9900 for details.

ORIGIN AND SUPPORT OF PANEL LIBRARY

The Panel Library code was written by David Tristram at NASA Ames Research Center in the late 1980's. David left NASA in 1989 and now works for Silicon Graphics International.

Creon Levit inspired some of the original design.

This manual was reviewed by (YOUR NAME HERE! JUST REPORT ERRORS AND OMISSIONS TO PAM: walatka@orv.nas.nasa.gov).

Abstract

The Panel Library provides a user-interface toolkit for the Silicon Graphics IRIS workstation family. It is used by programmers writing applications for the IRIS. The Panel Library allows the application user to point and click with the mouse rather than type input with the keyboard. Output is graphically displayed.

User-interfaces built using the Panel Library consist of "actuators" and "panels". Actuators are buttons, dials, sliders, or other mouse-sensitive symbols that are capable of being visible on the screen. The user clicks the mouse when the cursor is on an actuator, and thus selects the actuator. When a button is selected, it changes color, and the fact that it is selected is sent to the program, changing the value of the button's field. Other actuators can be manipulated by moving the mouse. When the user selects a slider, and drags the mouse while holding down the mouse button, the value associated with the slider is changed, and slider's appearance changes to reflect the new value. This all happens instantly and with very little effort on the part of the user (or the programmer).

Usually, only the left mouse button is used.

Panels are groups of actuators that occupy separate windows on the Silicon Graphics IRIS Workstations. Like all IRIS Graphics Library windows, panels can be stored as icons when not in use. Actuators also can be stowed.

The Panel Library offers a number of programmer functions that access two data structures: 1. the descriptor of panels, and 2. the descriptor of actuators.

This draft of this manual contains an introductory chapter with sample code and notes on initialization, connecting the panel library to the rest of your program, scripting, action functions. There are other chapters on programmer's functions, types of actuators, fields of the panel and actuator structures, global variables, and behavior functions.

Good luck!

Symbols, Conventions, and Keywords

actuator (lower case):	a dial or slider, etc., capable of appearing on the screen
Actuator(Iinitial capital):	the data structure that defines an actuator
panel(lower case):	a group of actuators, appearing as an IRIS window on the screen
Panel(Initial capital):	the data structure that defines a panel
lower case bold:	a field in one of the two data structures: Panel structure Actuator structure
PNL_SMALL CAPS:	manifest constants defined for the Panel Library
SMALL CAPS:	manifest constants
italics():	name of programmer's functions
italics:	global variable

Contents

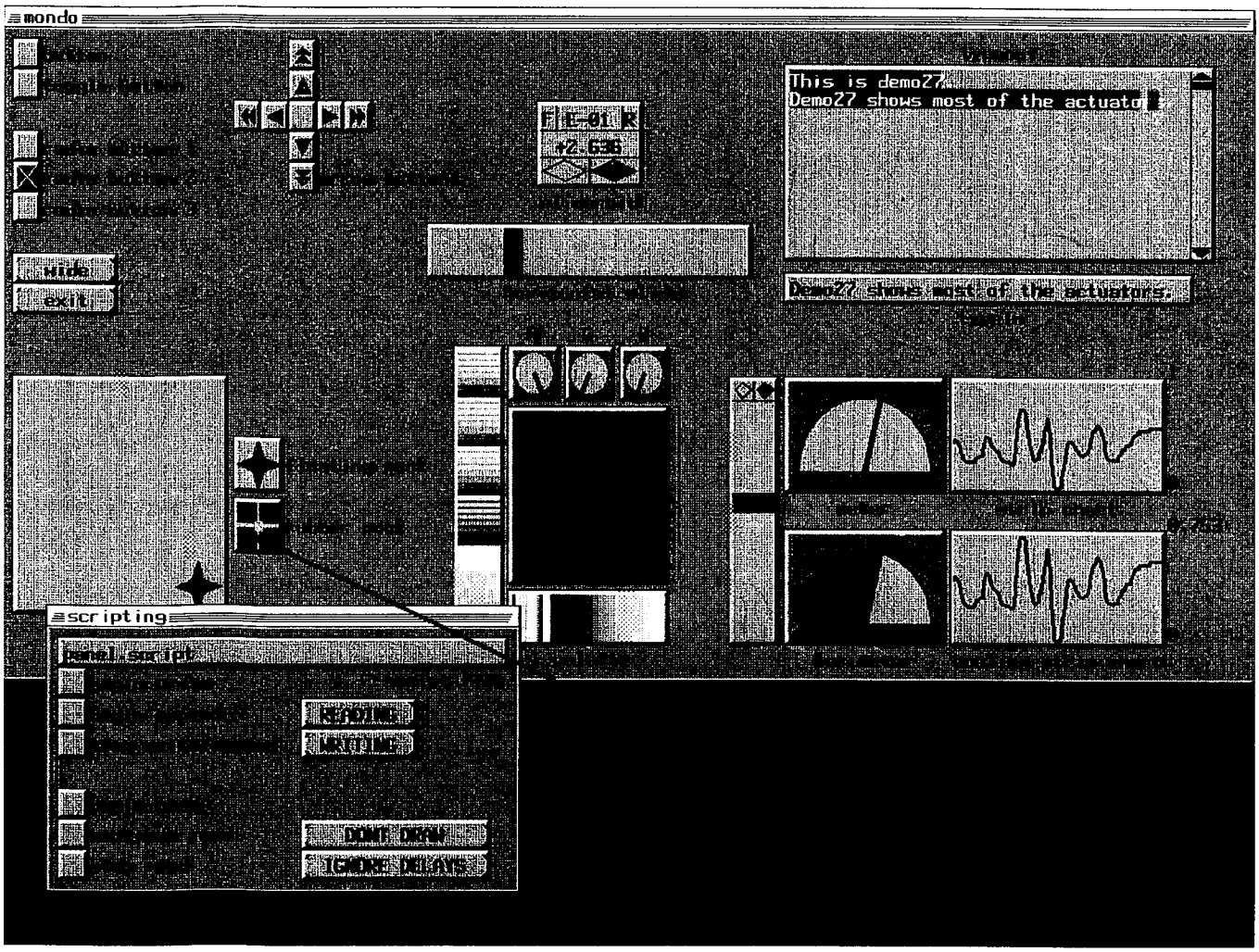

Figure 1. Demo 27—examples of the basic actuators. Two panels are shown: "Mondo" contains buttons, radio buttons, wide buttons, arrow buttons, sliders, a slideroid, typin, typeout, meter, bar meter, strip chart, scaling strip chart, palettes, dials, and pucks. "Scripting" contains buttons that activate scripting functions. The scripting panel is pre-defined in the Panel Library.

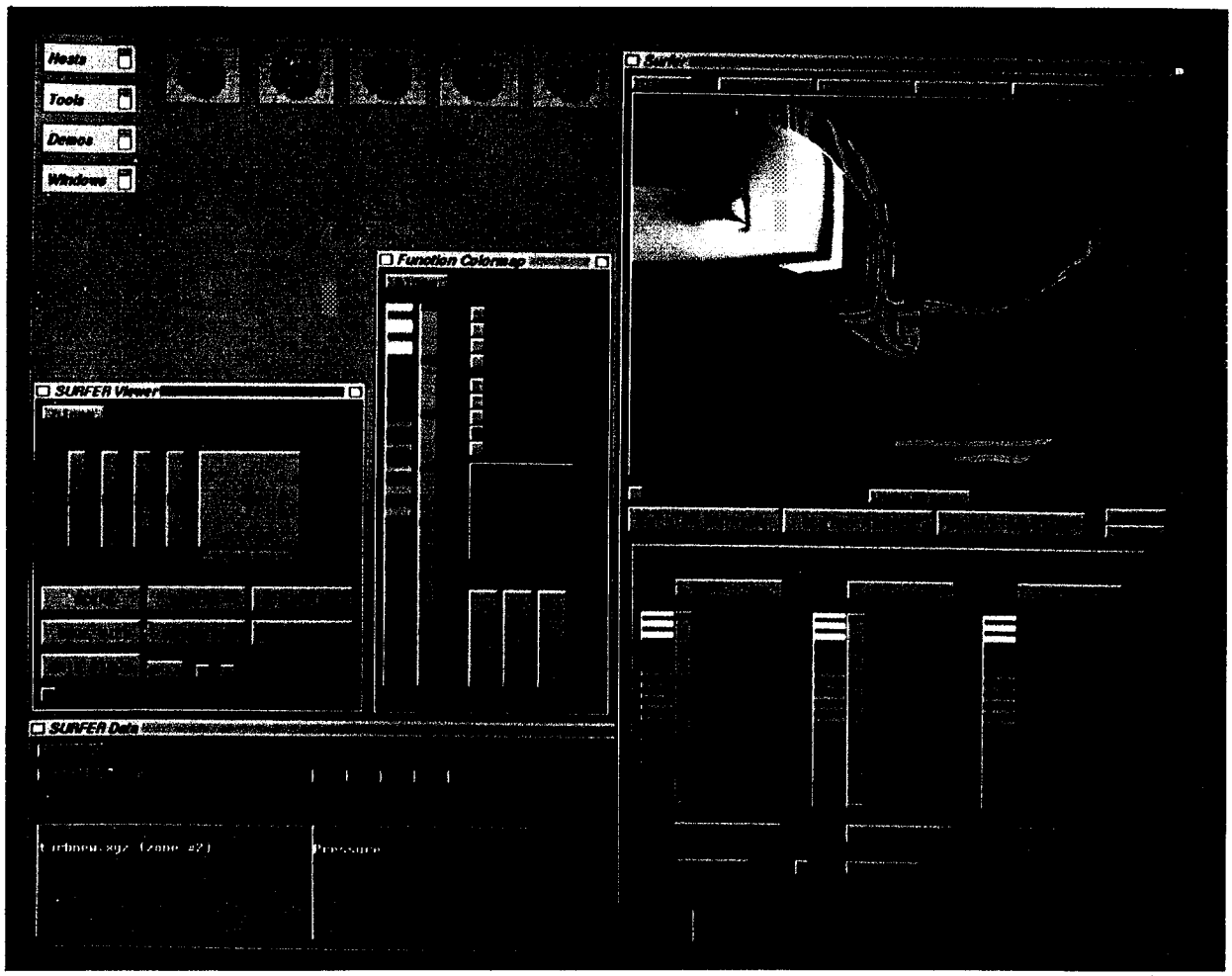

Figure 2. Example of panels and actuators used in a previous version of FAST. At the top-left are purple icons of stowed panels. At mid-left is a panel—SURFER Viewer—with wide-buttons, sliders, and a puck. In the middle, the Function Colormap panel has radio buttons, a wide-button, a multi-slider, two palettes, and three sliders. At bottom left, the SURFER Data panel has wide-buttons, buttons, and typeouts. The Surfer panel has pull-down menus, wide-buttons, a viewframe actuator, buttons, typeins (Top, Top, Max), and multi-sliders with palettes. See New Actuators for information on viewframes.

Chapter 1

Introduction

1.1 Interactive Visualization Using the Panel Library

The Panel Library is a toolkit for building interactive graphics applications on Silicon Graphics IRIS workstations. The Panel Library provides sliders, buttons, and assorted instruments that are used as graphic input/output devices. With the IRIS mouse, the application user can alter any variable in the graphics program, or fire off functions with a click on a button. The evolution of data values can be tracked with meters and strip charts, and dialog boxes with text processing typeins and typeouts can be built. You will be surprised at how easy it is to build sophisticated interactive user interfaces with the Panel Library.

Programmers can start using the Library quickly, basing their work on the more than forty demo programs shipped with the source. Creating actuators and control panels is easy, and requires calling only a few Library functions. After an actuator is created, any aspect of it can be modified by updating the corresponding field in its descriptor. The Library immediately and automatically updates the graphical representation of the actuator on the screen to match its new position, size, or value.

Our approach with the Library so far has been to develop a framework that supports an extensible number of different actuators. This way, new actuators or special versions of existing ones can be made part of the system to add new features to the Library. Then, we used the Library in the development of some of our research codes, like the interactive flow visualization applications FAST and ISOLEV, to find out what features and capabilities

we really neded. By refining the Library with real applications we have built a toolkit that is actually useful.

In the future we intend to add features and capabilities (and, yes, fix bugs) to meet NAS program requirements. The Panel Library is a big part of the major NAS flow analysis software effort FAST, and is being used at over fifty other research, academic and industrial sites around the world. As these projects progress, the Panel Library will be enhanced to keep pace with their needs.

1.2 This Manual and the Demonstration Programs

This reference manual is in a draft stage. It provides documentation of the library's functional interface, the actuator types, the use and meaning of the structure fields, the global variables used to control the library, and behavior functions. In this chapter it also gives some words on some basic concepts and design philosophy. However, it does not provide much high-level help for the new user of the library. New users are directed to the dozens of demonstration programs provided with the source code(D.demos). Each demo illustrates a new feature of the library, and, incidently, the order of the demos reflects its development history. The Panel Library is an extremely powerful and flexible toolkit, and there are usually a couple of ways to do what you want to do, so take some time when developing your application to consider the solution best for you.

1.3 Some Sample Code

To access the Panel Library, use an include statement as shown below.
The following is a code fragment that allocates two global variables to hold pointers to actuators, and defines a function that, when called, creates a panel, creates two sliders with specified labels, positions, initial values, and value ranges, adds the sliders to the panel, and returns a pointer to the panel.

```
#include <panel.h>

  /* globally accessible pointers to actuators */

Actuator *slider1, *slider2;

 /* call defpanel() before pnl_dopanel() to create a panel */
 /* containing two sliders. */

Panel
*defpanel()
{
  Panel *panel;

  panel=pnl_mkpanel();                /*create a panel structure */

  panel->label="position control";    /*give it a title bar string*/

  slider1=pnl_mkact(pnl_hslider);     /* create a horizontal slider */
  slider1->label="x position";        /* assign some of its properties */
  slider1->x=1.0;
  slider1->minval= -1.0;
  slider1->maxval=1.0;
  pnl_addact(slider1, panel);         /* place the slider on the panel */


  slider2=pnl_mkact(pnl_vslider);     /* create a vertical slider */
  slider2->label="y position";        /* assign some of its properties */
  slider2->minval= -1.0;
  slider2->maxval=1.0;
  pnl_addact(slider2, panel);        /* place the slider on the panel */

  return panel;
}
```

After creating the panel, the programmer places a call to *pnl_dopanel()* in his or her program's main loop to process input events and animate the controls on the screen.

```
main() {
    winopen("my.prog");
    defpanel();

    while(1)
 pnl_dopanel();
 draw_my_graphics();
    }
}
```

1.4 Initialization

The function *pnl_dopanel()* expects that the application has already initialized its own graphics window for data output if there is to be one. If there is to be no user data window, the application must still initialize the graphics subsytem with *noport()* and *winopen()*.

The first time *pnl_dopanel()* is called, these initialization steps are performed:

1. The Panel Library sets the RGB color stored in nine locations of the colormap. The indices used can be changed by setting the panels color variables before *pnl_dopanel()* is called. The application programmer can change the color of control panels by setting the RGB colors stored in these locations or by changing the indices in the color variables after initialization has been performed.

2. Each panel created previously with *pnl_mkpanel()* is then initialized. The limits (**minx**, **maxx**, **miny**, **maxy**) of the world space coordinates are calculated. This is the bounding box of all actuators in the panel, plus a small border value.

3. If the application programmer has not set the height or width (**h** and **w**) of the panel (in screen space), the library determines the panel's size based on the world space limits and the panel's scalefactor (**ppu** or pixels per unit).

4. If the programmer has not set the panel's screen space origin (**x** and **y**), the library places the panel to the right and aligned at the top of the application's data window, or below and aligned on the left of the previous panel if there is one. If a panel would extend off the screen on the bottom, the library attempts to place it to the right of the topmost, rightmost panel. If a panel extends off the right of the screen, the library offsets it down and to the right 20 pixels from the upper right of the data window.

5. A window for this panel is then created and constrained to maintain its aspect ratio. On IRIS 4D's, the window is placed in doublebuffered colormap mode. An orthographic viewing transformation mapping the panel's world space limits to the edges of its window is created and saved in **vobj**.

6. After all panels are initialized, the library queues the LEFTMOUSE, LEFTSHIFTKEY, RIGHTSHIFTKEY,CTRLKEY, (LEFTCTRLKEY and RIGHTCTRLKEY on the IRIS 4D's) devices and ties the MOUSEX and MOUSEY devices to the LEFTMOUSE device. If the workstation is running the NeWS (4sight) window manager, the WINFREEZE and WINTHAW devices are queued to allow the application to continue running when a panel has been iconified.

1.5 Connecting The Panel Library To The Rest Of Your Program

In any real Panel Library application, there must be some connection between the actuators that you create and the rest of your application. There are four basic ways of using the Panel Library in order to accomplish this goal:

1. using globals
2. using actionfuncs
3. using the return value of *pnl_dopanel*
4. using additional data.

Which of these is preferable depends on your specific needs—very often more than one might be used in a single program.

1.5.1 Global Actuators and Panels

If a given actuator is global, then it is possible for any procedure to examine any of its fields. A particularly useful one to examine is the **val** field of any actuator. For most actuators, mousing it will change the **val** field according to the mouse position and the type of the actuator. To give some common examples: a toggle button's **val** will be either 0 or 1, while a slider's **val** field will be a floating point number somewhere between that slider's **minval** and **maxval.**

In other cases, the data relevant to a particular actuator is more complicated that just a single floating point number. This is true for actuators like pucks and typeins. For them, there is a special macro PNL_ACCESS defined in panel.h to extract the relevant information.

The D.demos directory has many examples of using certain fields of global actuators to affect the rest of the program.

1.5.2 Action Functions

Panel Library actuators and panels have as fields three pointers to functions that are called when the user operates the mouse. These are called the action function pointers and are named **downfunc, activefunc,** and **upfunc.** The application programmer assigns the addresses of user-defined functions to these pointers. Two of them are called when the mouse button makes the transition from up to down or down to up, and the third is called whenever the mouse button is down.

These action functions (actionfuncs) are for the sole use of the application program. Every actuator and every panel has three actionfuncs; for the purposes of this discussion, we will focus on the actionfuncs of actuators only.

For most actuators [1], the **downfunc** is called when that actuator is first selected (i.e. when the left mouse button is pressed while the mouse cursor is on top of that actuator), the **activefunc** is called as long as the mouse button is held down, and the **upfunc** is called when the mouse button is released. Naturally, these functions only get called if the programmer has assigned (pointer to function) values to them.

[1] The most important exception is pnl_typein—the documentation for pnl_typein explains when it calls its actionfuncs.

In all cases, if an actionfunc is called, it is passed a single argument—a pointer to the actuator that was responsible for that actionfunc being called. That actionfunc can then examine any field of the actuator in order to perform its job. Among other things, this means that the same actionfunc can be used for different actuators, and that actuators with actionfuncs generally don't need to be globals.

Since actionfuncs get called from within *pnl_dopanel()*, the **downfunc** (for instance) will not get called as soon as the mouse button is pressed, but rather during the first call to *pnl_dopanel* AFTER the mouse button is pressed. While in most cases the actionfunc will be called almost instantly, if there is time-consuming processing happening in the main loop, there might be some noticeable delay.

The D.demos directory has many examples of using actionfuncs.

1.5.3 *Pnl_dopanel()* return value

Every call to *pnl_dopanel()* returns the actuator that was active during that call, or NULL. In some cases, the programmer might want to test this value to perform some action. In addition, a Panel Library global variable (*pnl_funcmode, pnl_justdown, pnl_ca*, etc) can be examined.

Note: in most cases using actionfuncs (described above) is more flexible and modular. Using *pnl_dopanel*'s return value should only be using in special circumstances, not as the default technique.

1.5.4 Using additional data

There are two separate mechanisms provided for associating additional data with a given actuator. Every actuator has a field named **u** (for user data) which is not used by the Panel Library at all. The programmer can use this field to hold immediate data, or cast it to be a pointer to an arbitrary function. The second method uses a macro called PNL_MKUSERACT to extend an actuator with additional fields.

Examples of both of these techniques can be found in the D.demos directory.

1.6 Signal Handling

The Panel Library provides a special actuator type for handling UNIX signal interrupts received from other processes. This actuator is called *pnl_signal*. One instance of *pnl_signal* is created for each signal that is to be caught. Handling functions for the signal may be attached to the actuator's **downfunc** and **activefunc** active functions. A pointer to the signal actuator is returned by the next call to *pnl_dopanel*. See the description of the *pnl_signal* actuator for more detailed information (not available in this draft).

1.7 Scripting

The Panel Library has the capability to save user interactions to a script file on disk for later replay. In all but a few cases, when reading from a script file, the user interface will behave exactly as though the user were operating the mouse. Scripting is controlled by five functions, *pnl_beginreadscript()*, *pnl_beginwritescript()*, *pnl_beginappendscript()*, *pnl_endreadscript()*, *and pnl_endwritescript()*. A predefined panel providing buttons activating these functions is provided by the library and may be added to any application by calling *check this name initscriptpanel()* when defining the application's panels. Scripts may be interpreted in a human readable form by using the *stoa* utility.

1.8 Script Files

Script files consist of three types of packets: mouse, delay, and state, which are identified by reserved tokens, followed by binary data.

Mouse packets are generated whenever an actuator is "active", which usually means that the user has selected an actuator with the mouse button, and is still holding the button down. Mouse packets consist of the PNL_MT_MOUSE token, the **id**'s of the current panel and actuator, the current world coordinates of the mouse in the current panel, and five bits of key state information, the values of **pnl_justup**, **pnl_justdown**, **pnl_mousedown**, **pnl_shiftkey**, and **pnl_controlkey**.

Delay packets are generated when a number of frames, or calls to *pnl_dopanel()* have occurred without writing a mouse or state packet. This allows the Panel Library to wait while reading a script until the specified number of frames

have gone by, keeping playback of scripts roughly time-synchronous with the recorded version. Delay packets consist of the PNL_MT_DELAY token and the number of fames since the last mouse or state packet.

State packets attempt to save all information regarding an actuator. A state packet consists of the PNL_MT_STATE token followed by a dump of an actuator descriptor. If there is actuator-specific data associated with the actuator, it follows.

Chapter 2

Programmer Functions

The programmer's functions are discussed in alphabetical order, except when grouped as in the following list. They are:

pnl_addact()
pnl_addpanel()
pnl_addsubact()
pnl_delact()
pnl_delpanel()
pnl_dopanel()
pnl_drawpanel()
pnl_endgroup()
pnl_fixact()
pnl_fixpanel()
pnl_mkact()
pnl_mkpanel()
pnl_mkuseract()

pnl_needredraw(), pnl_userredraw

initscriptfile(), pnl_beginreadscript(), pnl_beginwritescript(), pnl_beginappendscript(), pnl_endreadscript(), pnl_endwritescript(), pnl_dumpstate()

pnl_strwidth()

The most important are:
pnl_addact()
pnl_dopanel()
pnl_mkact()
pnl_mkpanel()

NAME

pnl_addact—add an actuator to a panel

SYNOPSIS

void pnl_addact(a, p)

Actuator *a;

Panel *p;

DESCRIPTION

Pnl_addact() adds an actuator to a particular panel, in these steps:

1. The pointer to the Actuator is saved in *pnl_table*, allowing subsequent reference to the actuator by its **id**.
2. The actuator is placed at the head of the panel p's actuator list (**al**).
3. The current value of a's **val** field is saved as its initval for subsequent reset operations.
4. The dimensions and offsets of a's label are calculated, based on the panel's scalefactor (**ppu**, i.e., pixels per unit).
5. The actuator's **addfunc** is called to perform any actuator-specific initialization.
6. If a is an automatic actuator, it is placed on the panel's **autolist**.
7. If a has a key equivalent, it is placed on the Panel Library's key list (*pnl_kl*) and its key device is queued.
8. The actuator is marked as needing to be redrawn.

SEE ALSO

pnl_addsubact()pnl_mkact()pnl_mkpanel()

NAME

pnl_addpanel—add a panel to a running application

SYNOPSIS

void pnl_addpanel(p)

Panel *p;

DESCRIPTION

Pnl_addpanel() is used to add a panel created with *pnl_mkpanel()* to the library's list of active panels. Panels that are created before the first call to *pnl_dopanel()* need not be explicitly added with *pnl_addpanel()*. *Pnl_addpanel()* is used to add panels that are created after the first call to *pnl_dopanel()*.

NAME

pnl_addsubact—add subactuator to actuator

SYNOPSIS

void pnl_addsubact(sa, a)

Actuator *sa;

Actuator *a;

DESCRIPTION

Pnl_addsubact() adds a subactuator, sa, to an actuator, a. Potentially, any actuator may be a subactuator of any other, but in practice certain actuators like menus, icons, and frames, are expected to have certain other actuators, like buttons and submenus, added to them. The Actuator to which the Subactuator is being added should already have been added to a Panel with *pnl_addact()*, or to another actuator with *pnl_addsubact()*. A subactuator, once added to another actuator, is not to be added to a panel with *pnl_addact()*, or vice versa. *Pnl_addsubact()* performs these steps:

1. *Pnl_addsubact()* saves a pointer to the Subactuator in *pnl_table*, allowing subsequent access to the Subactuator via its **id**.
2. The Subactuator is placed at the head of the Actuator's actuator list (**al**).
3. The value of sa's **val** field is stored as initval for subsequent resets.
4. The dimensions and offsets of the Subactuator's label are calculated based on the actuator's enclosing Panel's scale-factor (**ppu**).
5. The Subactuator's **addfunc** is called to perform any actuator-specific initialization.
6. The actuator's **addsubfunc** is then called to update any actuator-specific data structures it may have.
7. *Pnl_fixact()* is then called to allow the actuator to adjust its appearance (or other internal data structures) based on its new addition.

8. If sa is an automatic actuator it is placed on a's panel's **autolist.**
9. If sa has a key equivalent, it is placed on the Panel Library's key list (*pnl_kl*) and its key device is queued.

DIAGNOSTICS

Pnl_addsubact() prints a warning and exits if a has not been added to a panel or another actuator.

SEE ALSO

pnl_addact(),pnl_mkact()

NAME

pnl_delact()—delete an actuator and its subactuators

SYNOPSIS

void pnl_delact(a)

Actuator *a;

DESCRIPTION

Pnl_delact() removes an actuator a and all its subactuators (as found in its **al**) from the panel library's data structures. The actuator's entry in *pnl_table* is set to NULL, and then *pnl_delact()* is called for each of the subactuators in **al**. If a is a top-level actuator (originally added to a panel with *pnl_addact()*), it is removed from its panel's **al**. If a is a subactuator of some other actuator, it is removed from that actuator's **al** if it appears there. If a is a member of a group it is removed from it. If a has a *delfunc* it is then called. The default *delfunc* frees storage used for the actuator-specific data structure. A is deleted from its panel's **autolist** if it is there, and deleted from the libraries list of actuators with key equivalents *(pnl_kl)*, unqueueing its associated key device if it appears there. Finally, the space referenced by a is freed.

DIAGNOSTICS

warning: wanky group ring for act < *label* > when a cycle is detected in the **group** list.

SEE ALSO

pnl_delpanel()

NAME

pnl_delpanel()—delete a panel and its actuators

SYNOPSIS

```
void pnl_delpanel(p)
Panel *p;
```

DESCRIPTION

Pnl_delpanel() may be called at any time to delete a panel from the library. The actuators added to the panel are also deleted automatically using *pnl_delact()*. Since memory used by the panel structure is freed, a panel should not be referenced after it has been deleted.

SEE ALSO

pnl_delact()

NAME

pnl_dopanel—initialize and update panels

SYNOPSIS

Actuator *pnl_dopanel()

DESCRIPTION

Pnl_dopanel() performs the event processing and controls the display updates for all panels and actuators created for an application. *Pnl_dopanel()* is intended to appear once inside the application's main event loop.

1. The first time *pnl_dopanel()* is called, it creates the windows that are to contain the panels, and performs other initialization for panels (see Initialization).
2. *Pnl_dopanel()* then gets an "action" from the mouse or an event script (see Scripting). If an actuator has been selected, it is saved as the current actuator (available as *pnl_ca*), and *pnl_dopanel()* calls its **newvalfunc**, allowing the actuator to update its **val** field based on mouse clicks or movement.
3. *Pnl_dopanel()* then calls the **newvalfunc** of any "automatic" actuators (actuators that had automatic set to true when they appeared in an *pnl_addact()* call).
4. The **drawfunc** for each panel is called. The default **drawfunc** for panels fills the panel's window with the background color, and then calls the **drawfunc** of each actuator in its actuator list (**al**) whose **dirtycnt** is non-zero. The dirtycnt of these actuators is then decremented.
5. If an event script is being written, *pnl_dopanel()* writes the current actión to disk.
   On Iris 4D's, when a mouse button has just gone up, *pnl_dopanel()* attempts to process another action from the event queue and returns to step 2.

Normally *pnl_dopanel()* returns control to the calling program if no input events are available for processing. Setting *pnl_block* to TRUE will cause *pnl_dopanel()* to block until an event is read from the event queue. A caught signal will interrupt a blocked call to *pnl_dopanel()*. (See Signal Handling).

RETURN VALUE

Pnl_dopanel() returns a pointer to current actuator or NULL. *Pnl_dopanel()* may be made to always return NULL by means of the *pnl_dopanel_return_mode* global variable.

BUGS

Because actuator descriptors contain pointers to other data structures, loading state information from script files does not always work correctly, and can cause segmentation violations.

Setting a panel's **x** or **y** to zero will not have the desired effect of placing the window at the bottom or left hand edge of the screen because *pnl_dopanel()* uses zero as indication that the user has not expressed an interest in the panel's position and places it automatically according to the rules described in the Initialization section. **W** and **h** should also not be set to zero, but this is not as likely to occur intentionally.

SEE ALSO

pnl_block

NAME

pnl_drawpanel—draw all panels

SYNOPSIS

void pnl_drawpanel()

DESCRIPTION

Pnl_drawpanel() calls the **drawfunc** for every panel, which by default calls the **drawfunc** for every actuator whose **dirtycnt** is non-zero. If *pnl_dopanel()* has not been called yet, *pnl_drawpanel()* calls it to perform initialization.

RETURN VALUE

void

SEE ALSO

pnl_dopanel()

NAME

pnl_dumppanel—create file defining current panel configuration

SYNOPSIS

Boolean pnl_dumppanel()

DESCRIPTION

Pnl_dumppanel() is not really implemented and has been replaced by the panel editor. See the README section.

NAME

pnl_endgroup—signal end of group of actuators

SYNOPSIS

void pnl_endgroup(p)

Panel *p;

DESCRIPTION

Pnl_endgroup() is used to signal the end of a group of actuators. Some actuators, by their nature, form groups that must interact. For example, radio buttons, when selected, deselect all other radio buttons in their group. Radio buttons are automatically added to the current group when they are added to a panel. To have more than one group of radio buttons on a panel, use *pnl_endgroup()* to end the current group. Radio buttons added subsequently will appear in their own group.

SEE ALSO

pnl_radio_button

NAME

pnl_fixact—update actuator

SYNOPSIS

void pnl_fixact(a)

Actuator *a;

DESCRIPTION

Pnl_fixact() performs generic updating of an actuator's internal state to reflect changes that the application may have made to it. For example, *pnl_fixact()* should be called after setting an actuator's **val** field. This notifies the library that the actuator should be redrawn in its new position. Some actuators perform more extensive reorganization when "fixed". For example, Stripcharts copy the contents of their **val** field into their array of stored points, and delete the oldest value there. Typeouts insert the string pointed to by their **str** field into their text buffer.

RETURN VALUE

void

SEE ALSO

pnl_fixpanel()

NAME

pnl_fixpanel—update panel

SYNOPSIS

void pnl_fixpanel(p)

Panel *p;

DESCRIPTION

Pnl_fixpanel() causes the Panel Library to recalculate the size and location of a Panel based on any changes that may have been made to the visibility (**visible** field), iconification, size, or location of actuators in that panel. Explicit changes made to the location or size of the panel (**x**, **y**, **w** and **h**) should be followed by a call to *pnl_fixpanel()*.

SEE ALSO

pnl_fixact()

NAME

pnl_mkact—create Actuator descriptor

SYNOPSIS

Actuator *pnl_mkact(initfunc)

void initfunc();

DESCRIPTION

Pnl_mkact() performs the following operations:

1. *Pnl_mkact()* allocates space for an actuator descriptor.
2. A number of fields in the descriptor are initialized to their default values.
3. Functions providing default behaviors for some of the function pointers (**initfunc, pickfunc, dumpfunc, and loadfunc**) in the descriptor are assigned, as is the actuator's default label position (**labeltype**).
4. *Pnl_mkact()* then calls the function whose address is provided by its argument **initfunc**, which customizes the fields of the descriptor for the specific type of actuator it is to represent. Because this customization function is provided as an argument, a programmer may add new types of actuators to his application without changing the Panel Library itself.

RETURN VALUE

returns pointer to created Actuator.

SEE ALSO

pnl_addact()pnl_mkpanel()

NAME

pnl_mkpanel—create Panel descriptor

SYNOPSIS

void pnl_mkpanel()

DESCRIPTION

Pnl_mkpanel() performs the following operations:

1. *Pnl_mkpanel()* allocates space for a Panel descriptor.
2. The descriptor is placed on the Panel Library's list of panels (*pnl_pl*), and its address is stored in *pnl_table*, allowing subsequent reference to the panel to be made by its **id**.
3. A number of fields in the descriptor are initialized to their default values. Among these are the location and size of the panel, fields **x**, **y**, **w**, and **h**, which are initialized to zero. If the application programmer changes these values, the panel library will not attempt to automatically place and size the panels.

RETURN VALUE

returns pointer to the new Panel structure.

SEE ALSO

pnl_dopanel()

NAME

pnl_mkuseract; PNL_MKUSERACT—create actuator with extra storage for user data

SYNOPSIS

Actuator *pnl_mkuseract(size, initfunc)

int size;

void (* initfunc)();

struct useract *PNL_MKUSERACT (useract, initfunc)

useract- name of user structure

void (* initfunc)();

DESCRIPTION

Pnl_mkuseract() returns a pointer to an Actuator structure that has extra storage following contiguously. It is intended to allow the programmer to build "useracts" that appear like normal actuators to the Panel Library, but when accessed by user code reveal extra user fields. *PNL_MKUSERACT()* is a macro that conveniently calculates the size of the useract and casts the pointer returned by *pnl_mkuseract()* to that type.

RETURN VALUE

returns pointer to newly created user data structure.

DIAGNOSTICS

Warning: size in *pnl_mkuseract()* smaller than an actuator.

SEE ALSO

pnl_mkact(), and the demo program in src/D.demos/eric.c

name

pnl_needredraw; pnl_userredraw—save data window REDRAW events

SYNOPSIS

void pnl_needredraw()

short pnl_userredraw()

DESCRIPTION

Pnl_needredraw() is used to express an interest in REDRAW events for the application's data window(s). After *needredraw()* is called, the library will not discard REDRAW events that are for non-panel windows, but will instead requeue any such event if it is encountered while processing mouse events. The programmer who needs to process REDRAW events for the application's data window(s) then uses *pnl_userredraw()* to check whether the next event in the event queue is such a REDRAW. *Pnl_userredraw()* is intended to appear once inside the application's main event loop.

RETURN VALUE

returns the identifier of the user window receiving the REDRAW.

NAME

initscriptfile; pnl_beginreadscript; pnl_beginwritescript; pnl_beginappendscript; pnl_endreadscript; pnl_endwritescript; pnl_dumpstate—control creation and use of script files

SYNOPSIS

void initscriptfile()

Boolean pnl_beginreadscript(name)
char *name;

Boolean pnl_beginwritescript(name)
char *name;

Boolean pnl_beginappendscript(name)
char *name;

Boolean pnl_endreadscript(name)
char *name;

Boolean pnl_endwritescript(name)
char *name;

void pnl_dumpstate()

DESCRIPTION

These routines control the generation and consumption of script files (see Scripting).

Initscriptfile() creates a predefined panel to control the following scriptfile functions.

If the library is not currently writing a script, *pnl_beginreadscript()* opens for reading the file whose pathname is specified as a parameter. *Pnl_beginreadscript()* sets the file pointer back to the beginning of the file. The library then begins to read packets from the script, processing mouse events as though they were performed by the user.

If the library is not currently reading a script, *pnl_beginwritescript()* opens for writing the file whose pathname is specified as a parameter. If another file is already open, it is closed. If the file already exists, it is overwritten. Subsequent mouse events are written to the file as they are processed by the library.

Pnl_beginappendscript() behaves like *pnl_beginwritescript()* but instead of overwriting an existing file, new mouse events are appended to it. In the special case of the input scriptfile name being the same as the file specified as a parameter to *pnl_beginappendscript()*, new mouse events are added after the last event read from the input script. This is intended to allow primitive editing of scriptfiles.

Pnl_endreadscript() discontinues reading of a scriptfile, if it is currently happening. The scriptfile remains open with its file pointer located after the last event read.

Pnl_endwritescript() discontinues writing of a scriptfile if it is currently happening. The scriptfile is closed.

Pnl_dumpstate() dumps the current state of all actuators on all panels to the current scriptfile, opening it if needed. Later reading of the state information will cause the actuators to assume the state they had when dumped.

RETURN VALUE

these functions return TRUE upon successful completion, or FALSE otherwise.

BUGS

pnl_dumpstate(): because actuator descriptors contain pointers to other data structures, loading state information from script files does not always work correctly, and can cause segmentation violations.

NAME

pnl_strwidth—return width of string in panel coordinates

SYNOPSIS

pnl_strwidth(p, s)

Panel *p;

char *s;

DESCRIPTION

Pnl_strwidth() returns the width of a string in panel world space coordinates, much as the Iris Graphics Library call *strwidth()* does, but bases its computations on the character width of the standard font and the panel's **ppu** (pixels per unit) field. This is to allow the calculation of string widths before a window for the panel has been created.

RETURN VALUE

returns the length of the string s in world coordinates and 0 for the NULL string.

Chapter 3

Actuators

3.1 Overview of Actuator Types

The Panel Library provides the following types of actuators.

- Buttons
- Sliders
- Palettes
- Pucks
- Other Valuators
  - Dial
  - Slideroid
- Meters
- Strip Charts
- Text Manipulators
- Mouse
- Compound Actuators
  - Multislider
  - Grouping Actuators
    - Frame
    - Icon
    - Cycle
    - Scroll
  - Menus
- Signal
- New Actuators
  - Viewframe

These actuator types are introduced briefly below, and described in detail in the section following.

3.1.1 Buttons

Buttons are the most basic actuator provided by the Panel Library. They turn on, they turn off.

3.1.2 Sliders

Sliders are continuous one-dimensional controllers. The current value of the slider is controlled by and reflected by the position of a slider bar within a bounding rectangular region. In general, the user changes the value of the slider by mousing (depressing the mouse button) within the rectangular region. The bar jumps to the mouse location and then follows subsequent mouse motion until the mouse button is released. If the user runs the bar up against the "stops", mouse motion in the opposite direction results in bar motion immediately; in this way the bar will no longer line up with the mouse, but will continue to follow its motion.

3.1.3 Palettes

Palettes operate identically to sliders, but have no bar to indicate their current value. Their background is a palette of adjacent colors from the colormap.

3.1.4 Pucks

Pucks are continuous two dimensional controllers. The current value of a puck is controlled by and reflected by the position of a puck icon within a rectangular (square by default) bounding region, or, in the case of floating and rubber pucks, its position relative to a small "home" rectangle.

3.1.5 Other Valuators

Dial

The dial is a continuous rotary controller. The current value of the dial is controlled by and reflected by the angular position of a mark which moves around the dial's circular face. The face rests on a rectangular background.

The user changes the value of the dial by clicking within the background. The dial mark snaps to the line connecting the center of the dial and the mouse position, and then follows the mouse as the user moves it.

Slideroid
A slideroid is a one-dimensional continuous controller. It provides differential and fine control over a numeric readout. Slideroid has no graphical representation of its value; instead of moving a puck or slider bar, the user mouses in small control regions to alter the actuator's value.

3.1.6 Meters

Meters are output-only actuators designed to resemble mechanical analog electrical measuring devices.

3.1.7 Strip Charts

Strip charts are rectangular plotting regions used to display a history of data values. As data values are added to the strip chart, they are plotted along the ordinate. The position along the abscissa reflects the order in which the data values are presented to the strip chart. Strip charts automatically scroll to display the most recently added data.

3.1.8 Text Manipulators

The Panel Library provides actuators to display and read text. Typein gets text from the user, Typeouts display and allows selection of multi-line text buffers in a scrolling window. Labels display short text strings in a fixed position in a window.

3.1.9 Mouse

The Panel Library provides a special invisible actuator that returns the mouse position and allows the programmer to attach functions to the mouse buttons.

3.1.10 Compound Actuators

The Panel Library supports compound actuators, or actuators to which sub-actuators have been added. Programmers may build their own hierarchical

actuators using this capability, adding functionality by attaching one kind of working actuator under another that has some other desirable property. The grouping actuators are used this way. The Panel Library also provides some compound actuators that have already been assembled and may be operated as a single unit like the simple actuators described above.

Multislider

The multislider resembles a slider, with the ability to contain additional slider bars. The bars are manipulated independently and may have their own action functions.

Grouping Actuators

The grouping actuators allow the programmer to treat a number of actuators as a single actuator, and to modify the behavior of existing actuators.

Frame

A frame is the basic grouping actuator. Frames are used to impose a new viewing transformation on subactuators. Actuators drawn inside a frame may have their own scalefactor and origin. Manipulations applied to the frame are passed on to subactuators, for example, it the frame is made invisible, all subactuators disappear as well; if the frame is moved, its contents move, or if the frame is made unselectable, no actuator inside it may be selected. Frames are used instead of panels when a separate IRIS window is not desired.

Icon

An icon takes a single actuator and adds a momentary iconification behavior to it. That is, the icon appears on a panel as a small rectangle until selected with the mouse, whereupon it changes into its other visible form, to wit, that of its subactuator.

Cycle

A cycle displays one of a number of actuators that have been added to it. The user may click buttons to cycle left or right through the list of subactuators.

Scroll

A scroll is a grouping actuator that provides a means for scrolling back and forth over the area of a large group of actuators. Actuators are added to scrolls just as they are to frames, but the scroll has a fixed size, normally smaller than the frame required to display full-sized versions of the subactuators would be. The scroll displays a portion of the larger "frame" thus

created and the user may operate sliders to scroll back and forth to reveal other parts of the control panel.

Menus
The Panel Library menu actuator is a region of the screen that contains subactuators, much as a frame does. However, inside a menu, subactuators become active just by dragging (moving with the mouse button down) the mouse over them. Pop-up menus that spring from an on-screen icon are provided. Programmers can use submenu actuators to build nested menus.

3.1.11 New Actuators

Viewframe
Viewframe—refer to Figure 2—is available and supported, but not yet documented, except in viewframe.c on the source tape.

3.2 Actuator Descriptions

Table of Contents

Pucks

- pnl_puck
- pnl_floating_puck
- pnl_rubber_puck

Other Valuators

- pnl_dial
- pnl_slideroid

Meters

- pnl_meter
- pnl_analog_meter
- pnl_analog_bar

Strip Charts

- pnl_strip_chart
- pnl_scale_chart

Text Manipulators

- pnl_typein
- pnl_typeout
- pnl_label

Mouse

- pnl_mouse

Compound Actuators

- Multislider
  - pnl_multislider
  - pnl_multislider_bar
  - pnl_multislider_open_bar
- Grouping Actuators

pnl_frame
pnl_icon
pnl_cycle
pnl_scroll

Menus

pnl_menu
pnl_icon_menu
pnl_sub_menu
pnl_menu_item

New Actuators

Viewframe

3.3 Buttons

Buttons are the most basic actuator provided by the Panel Library. They turn on, they turn off.

NAME

pnl_button

DESCRIPTION

pnl_button is intended to operate as a simple momentary push-button

VALUE

val is set to **maxval** if the mouse button is down, **minval** otherwise.

APPEARANCE

pnl_button appears as a non-highlighted rectangle if **val** equals **minval**, and is highlighted otherwise.

ADDING OPERATIONS

none

FIXING OPERATIONS

none

NAME

pnl_toggle_button

VALUE

val alternates between minval and maxval upon mouse clicks.

APPEARANCE

pnl_toggle_button **appears as a** *pnl_button* **when its val is minval, and is highlighted with a cross from corner to corner otherwise.**

ADDING OPERATIONS

none

FIXING OPERATIONS

none

NAME

pnl_wide_button

VALUE

val is set to **maxval** if the mouse button is down, **minval** otherwise.

APPEARANCE

Pnl_wide_button appears as a wide non-highlighted rectangle with its label centered upon it if **val** equals **minval**. Otherwise, *pnl_wide_button* becomes highlighted, with its label drawn in inverse.

ADDING OPERATIONS

Pnl_wide_button increases its width based on the length of its label if necessary.

FIXING OPERATIONS

Pnl_wide_button increases its width based on the width of its label (as calculated by *labeldimensions()*) if necessary.

NAME

pnl_radio_button

VALUE

val is set to **maxval** (1.0) when *pnl_radio_button* is clicked upon, at which time any other radio buttons in its group is set to **minval**(0.0).

ADDING OPERATIONS

Pnl_radio_button is added to the panel's current group.

FIXING OPERATIONS

none

NAME

pnl_left_arrow_button

pnl_right_arrow_button

pnl_up_arrow_button

pnl_down_arrow_button

pnl_left_double_arrow_button

pnl_right_double_arrow_button

pnl_up_double_arrow_button

pnl_down_double_arrow_button

VALUE

val is set to **maxval** (1.0) if the mouse button is down, **minval** (0.0) otherwise.

APPEARANCE

These button's appear as non-highlighted rectangles containing a triangle or two pointing in some direction if **val** is zero, and are highlighted with the triangles in an inverse color otherwise.

ADDING OPERATIONS

none

FIXING OPERATIONS

none

BUGS

These should be implemented as "picture buttons" that take polygons to draw on their surfaces.

3.4 Sliders

Sliders are continuous one dimensional controllers. The current value of the slider is controlled by and reflected by the position of a slider bar within a bounding rectangular region. In general, the user changes the value of the slider by mousing (depressing the mouse button) within the rectangular region. The bar jumps to the mouse location and then follows subsequent mouse motion until the mouse button is released. If the user runs the bar up against the "stops", mouse motion in the opposite direction results in bar motion immediately; in this way the bar will no longer line up with the mouse, but will continue to follow its motion.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int mode;
  float finefactor;
  float valsave;
  Coord wsave;
  Coord bh;
} Slider;
```

mode:	current slider mode
finefactor:	ratio of slider bar motion to mouse motion when in fine control mode. Default value is PNL_FINE_CONTROL_FACTOR (1/20).
differentialfactor:	ratio of rate of slider bar movement to mouse motion for differential sliders in differential mode. Default value is PNL_DIFFERENTIAL_FACTOR (1/10).
valsave:	temp store for value between calls (internal)
wsave:	temp store for slider bar position.
bh:	height of slider bar. Default value is PNL_SLIDER_BAR_HEIGHT (PNL_SLIDER_HEIGHT/20).

MODES

These modes may be or'ed together.

```
#define PNL_SM_NORMAL 0x0
#define PNL_SM_DIFFERENTIAL 0x1
#define PNL_SM_FINE_CONTROL 0x2
#define PNL_SM_NOSNAP 0x4
```

PNL_SM_NORMAL:

No special modes.

PNL_SM_FINE_CONTROL:

All sliders have normal and fine-control modes. Fine control is selected by depressing the control key on the keyboard. Fine control mode implies PNL_SM_NOSNAP behavior, but does not explicitly set the PNL_SM_NOSNAP bit in mode.

PNL_SM_NOSNAP:

All sliders have no-snap mode to avoid moving the slider bar when first clicking on the slider. Set no-snap by or'ing PNL_SM_NOSNAP with mode.

PNL_SM_DIFFERENTIAL:

Indicates differential mode, see *pnl_dvslider, pnl_dhslider.*

BUGS

Fine control can not be selected by setting a slider's mode field to PNL_SM_FINE_CONTROL.

NAME

pnl_slider

pnl_vslider

pnl_hslider

pnl_filled_slider

pnl_filled_vslider

pnl_filled_hslider

Pnl_vslider is a vertical slider, and *pnl_hslider* operates from side to side (horizontally). *Pnl_slider* is a synonym for *pnl_vslider*. The filled sliders operate identically, and have a different appearance.

ACTUATOR-SPECIFIC DATA

See *pnl_slider*.

VALUE

The value of the slider is the value of the endpoints, set by **minval** and **maxval**, linearly interpolated by the position of the slider bar within the bounding rectangle of the slider.

APPEARANCE

These sliders are long rectangles containing a bar running across them. The bar slides up and down to reflect the slider's changing value. No highlighting is made when the slider is selected. Filled sliders draw the background of the slider on the low side of the slider bar is a contrasting color, providing a stronger (but anisotropic) visual cue for position.

ADDING OPERATIONS

none

FIXING OPERATIONS

none

NAME

pnl_dvslider

pnl_dhslider

These are "differential" sliders. *Pnl_dvslider* moves vertically, *pnl_dhslider* horizontally. At the high end of these sliders there are two additional mouse sensitive regions. One is for fine control and the other is for differential or motorized control. Dragging after clicking in the fine control region moves the slider bar in the ratio of finefactor to the mouse motion. Dragging after clicking in the differential region moves the slider bar at a rate proportional to the mouse motion.

VALUE

The value of the slider is the value of the endpoints, set by **minval** and **maxval**, linearly interpolated by the position of the slider bar within the bounding rectangle of the slider. The value can also be set by using the fine control and differential control regions of the slider (see above).

APPEARANCE

The differential sliders look like *pnl_slider*, with two additional regions at the high end. These regions are marked with diamond shapes, one open, one filled. The filled diamond marks the fine control region, and the open one the differential. Clicking on these regions highlights them and draws the diamond in an inverse color.

ADDING OPERATIONS

none

FIXING OPERATIONS

none

3.5 Palettes

Palettes operate identically to sliders, but have no bar to indicate their current value. Their background is a palette of adjacent colors from the colormap.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int mode;
  float finefactor;
  float valsave;
  Coord wsave;
  Coord bh;
} Palette;
```

mode:	current palette mode
finefactor:	ratio of value motion to mouse motion when in fine control mode. Default value is PNL_FINE_CONTROL_FACTOR (1/20).
differentialfactor:	unused
valsave:	temp store for VALUE between calls (internal)
wsave:	temp store for mouse position.
bh:	unused

note: the Palette typedef is actually defined in terms of the Slider typedef; that detail is eliminated from the declaration given here for clarity, see panel.h for actual declaration.

MODES

These modes may be or'ed together.

```
#define PNL_SM_NORMAL 0x0
#define PNL_SM_DIFFERENTIAL 0x1
#define PNL_SM_FINE_CONTROL 0x2
#define PNL_SM_NOSNAP 0x4
```

PNL_SM_NORMAL: No special modes.

PNL_SM_FINE_CONTROL: Palettes have normal and fine-control modes. Fine control is selected by depressing the control key on

the keyboard. Fine control inhibits setting of output value at mouse down (like NOSNAP mode).

PNL_SM_NOSNAP: Not Implemented (except as with fine control).

PNL_SM_DIFFERENTIAL: Not Implemented.

BUGS

Fine control can not be selected by setting a palette's mode field to PNL_SM_FINE_CONTROL.

NAME

pnl_palette

pnl_vpalette

pnl_hpalette

Pnl_vpalette is a vertical palette, and *pnl_hpalette* operates from side to side (horizontally). *Pnl_palette* is a synonym for *pnl_vpalette*.

ACTUATOR-SPECIFIC DATA

???

VALUE

The value of the palette is the value of the endpoints, set by **minval** and **maxval**, linearly interpolated by the position of the mouse within the bounding rectangle of the palette.

APPEARANCE

These sliders are long rectangles filled with a contiguous range of colors taken from the colormap. The indices of the colors at the ends of the palette correspond to the palette's **minval** and **maxval**. There is no indication of the current value or selection status of the palette.

ADDING OPERATIONS

none

FIXING OPERATIONS

none

3.6 Pucks

Pucks are continuous two dimensional controllers. The current value of a puck is controlled by and reflected by the position of a puck icon within a rectangular (square by default) bounding region, or, in the case of floating and rubber pucks, its position relative to a small "home" rectangle.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  float x, y;
} Puck;
```

x: the current x position of the puck.
y: the current y position of the puck.

note: the Puck typedef is actually defined in terms of the Point typedef; that detail is eliminated from the declaration given here for clarity, see panel.h for actual declaration.

BUGS

Pucks do not have a fine-control mode.

Access to a puck's x and y values must be made through the non-orthogonal PNL_ACCESS macro, or by using a specially declared pointer. This makes them harder to use, and different from, other actuators that just return one value through their val field.

Pucks have the same output scaling in both x and y.

NAME

pnl_puck

Pnl_puck **constrains a puck icon to a rectangular region of the screen.**

VALUE

The x and y values of the pnl_puck are the min and max values, linearly interpolated by the position of the puck icon relative to the left-right and bottom-top edges of the bounding region.

APPEARANCE

Pnl_puck **is a large square region containing a puck icon resembling a compass rose. No highlighting is made for selected *pnl_puck*.**

ADDING OPERATIONS

none

FIXING OPERATIONS

none

BUGS

Non-square bounding regions for *pnl_puck* give non-uniform output value scaling.

NAME

pnl_floating_puck

pnl_rubber_puck

Pnl_floating_puck uses a small rectangle as a home position for the puck shape, and allows the user to move the puck all over the screen. Pnl_rubber_puck is similar to **pnl_floating puck**, but draws a rubber band line back to the home position.

VALUE

The x and y values of pnl_floating_puck and pnl_rubber_puck are proportional to the distance the floating puck icon has been moved from the home position. When first clicked upon, these puck icons have not moved and thus their x and y values are zero.

APPEARANCE

Pnl_floating_puck is a puck icon shape (resembling a compass rose) on a small square region just large enough to contain it. When clicked upon, the background is highlighted, the icon is drawn in a contrasting color and a copy of the icon shape is also drawn in the workstation's overlay bitplanes. This overlay image moves to follow the mouse position. The normal mouse cursor is shut off. Pnl_rubber_puck uses a different shape (resembling a cross) for the puck icon, and in addition to the overlay image, draws a rubber band line from the image back to the home position, providing a stronger visual cue as to which puck is active.

ADDING OPERATIONS

none

FIXING OPERATIONS

none

3.7 Other Valuators

pnl_dial

The dial is a continuous rotary controller. The current value of the dial is controlled by and reflected by the angular position of a mark which moves around the dial's circular face. The face rests on a rectangular background. The user changes the value of the dial by clicking within the background. The dial mark snaps to the line connecting the center of the dial an the mouse position, and then follows the mouse as the user moves it.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int mode;
  float finefactor;
  float valsave;
  Coord wsave;
  float winds;
} Dial;
```

mode:	current dial mode
finefactor:	ratio of dial bar motion to mouse motion when in fine control mode. Default value is PNL_FINE_CONTROL_FACTOR (1/20).
valsave:	temp store for value between calls (internal)
wsave:	temp store for dial mark position.
winds:	number of full (360 degree) rotations of the dial corresponding to full range output. Default value is PNL_DIAL_WINDS (0.88).

MODES

These modes may be or'ed together.

```
#define PNL_SM_NORMAL 0x0
#define PNL_SM_DIFFERENTIAL 0x1
#define PNL_SM_FINE_CONTROL 0x2
#define PNL_SM_NOSNAP 0x4
```

PNL_SM_NORMAL: No special modes.

PNL_SM_FINE_CONTROL: Palettes have normal and fine-control modes. Fine control is selected by depressing the control key on the keyboard. Fine control inhibits setting of output value at mouse down (like NOSNAP mode).

PNL_SM_NOSNAP: Not Implemented (except as with fine control).

PNL_SM_DIFFERENTIAL: Not Implemented.

VALUE

The value of the dial is the **minval** and **maxval** fields linearly interpolated by the angular position of the dial mark. When the mouse button goes down, the dial mark lines up with the ray running from the center of the dial to the mouse position. The mouse may be moved far from the dial to achieve finer control over the angular position of the dial mark (and hence, of the dial's output value). Coupled with fine control mode, a dial can be used this way to adjust a variable to one part in 10000. After running up against the "stops", rotation in the opposite direction results in dial motion immediately; in this way the dial mark may no longer point at the mouse, but the dial will rotate as expected.

APPEARANCE

Pnl_dial is a circle resting on a rectangular background. A thin rectangle running from the center to the edge of the circle marks the dial's position. The location of the base positions, or end stops, of the dial are determined by the number of "winds" it takes to run over its full range. The middle of the range always falls at the top of the dial. If winds is greater than two, there will be other values corresponding to the mark being straight up as well. *Pnl_dial* does not highlight to reflect selection status.

ADDING OPERATIONS

none

FIXING OPERATIONS

none

BUGS

Fine control can not be selected by setting a dial's mode field to PNL_SM_FINE_CONTROL.

If an application is updating the screen at a low frame rate, *pnl_dial* can get confused about how far the mouse has moved. Clicking and rapidly moving the mouse down or off to the side will probably not work, and, if the mouse goes past "zero" degrees (straight down), the dial may not process that correctly, and jump to an unexpected position.

NAME

pnl_slideroid

Pnl_slideroid is a one-dimensional continuous controller. It provides differential and fine control over a numeric readout. *Pnl_slideroid* has no graphical representation of its value; instead of moving a puck or slider bar, the user mouses in small control regions to alter the actuator's value. The control regions are strictly analogous to those provided by *pnl_dvslider* and *pnl_dhslider*. *Pnl_slideroid* has an explicit fine control mode, which decreases sensitivity to mouse motion on both inputs. A reset button is provided to set pnl_slideroid's value to its initval. Another floating point variable may be reset to an arbitrary value at this time as well.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int mode;
  Boolean finemode, resetmode;
  float *resettarget, resetval;
  float valsave;
  Coord wsave;
} Slideroid;
```

mode:	current slideroid mode.
finemode:	fine control mode.
resetmode:	reset mode (TRUE ==> reset).
resettarget:	address of variable to reset when resetting actuator.
resetval:	value to assign to contents of resettarget.
valsave:	temp store for value between calls (internal)
wsave:	temp store for mouse position (internal).

MODES

There are no programmer-accessible predefined constants for slideroid modes. This is a bug. Internally they are very similar to those for sliders, see slideroid.c.

VALUE

Pnl_slideroid has two mouse sensitive control regions. One is for fine control and the other is for differential or motorized control. Dragging after clicking in the fine control region changes the output value of *pnl_slideroid* in the ratio of finefactor to the mouse motion. Dragging after clicking in the differential region increments the value at a rate proportional to the mouse motion. Two other regions set fine control mode and reset the actuator. In fine control mode, sensitivity to mouse motion in all control operations is reduced.

APPEARANCE

Pnl_slideroid is a rectangular region containing a floating point readout of its value, a separate exponent field to display an appropriate power of ten, two labeled areas, "F" and "R" for fine control mode and reset operations, and two control areas marked with diamond shaped icons, one open, one filled. The filled diamond marks the fine control region, and the open one the differential. Clicking on these regions highlights them and draws the diamond in an inverse color. The labeled fields are also highlighted when active.

ADDING OPERATIONS

none

FIXING OPERATIONS

none

3.8 Meters

Meters are output-only actuators designed to resemble mechanical analog electrical measuring devices.

NAME

pnl_meter

pnl_analog_meter

pnl_analog_bar

Pnl_meter is a synonym for*pnl_analog_meter*. *Pnl_analog_bar* operates identically but has a slightly different appearance.

VALUE

These actuators are insensitive to mouse input.

APPEARANCE

Pnl_analog_meter is a semicircular region on a rectangular background. A thin rectangle represents the meter's needle, which swings back and forth from center of the bottom of the semicircle.

ADDING OPERATIONS

none

FIXING OPERATIONS

none

3.9 Strip Charts

Strip charts are rectangular plotting regions used to display a history of data values. As data values are added to the strip chart, they are plotted along the ordinate. The position along the abscissa reflects the order in which the data values are presented to the strip chart. Strip charts automatically scroll to display the most recently added data.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int firstpt, lastpt;
  Boolean Bind\_Low, Bind\_High;
  float *y;
  Actuator *lowlabel, *highlabel;
} Stripchart;
```

firstpt:	index in storage array of first point to be plotted, this is the oldest data value.
lastpt:	index in storage array of last point to be plotted, this is the newest data value.
Bind_Low:	when false, allows the low data limit to auto-range.
Bind_High:	when false, allows the high data limit to auto-range.
y:	pointer to data storage array.
lowlabel:	label actuator used to display the low data limit.
highlabel:	label actuator used to display the high data limit.

NAME

pnl_strip_chart

pnl_scale_chart

Pnl_strip_chart **is the basic strip chart,** ***pnl_scale_chart*** **automatically adjusts its high and low limits to maximize vertical resolution. Their fixfuncs take the value of their val field and put it into their internal data storage array, scrolling their display if necessary to plot it.**

VALUE

Strip charts are output-only actuators. The val field is used to write data into the strip chart, see fixing operations below.

APPEARANCE

Strip charts are light rectangles with data values plotted across them using a heavy dark line. There are labels indicating the data values corresponding to the bottom and top of the rectangle.

ADDING OPERATIONS

When added to a panel, ***pnl_strip_chart*** **allocates space for PNL_STRIP_CHART_NPTS (default 50) data values. It also creates the label actuators that will serve as the data limit labels, and adds them to itself, thereby placing them on its actuator list (AL).**

FIXING OPERATIONS

Fixing ***pnl_strip_chart*** **causes the current value of val to be copied into the strip chart's data value store. If there are PNL_STRIP_CHART_NPTS already stored, the oldest is discarded.**

BUGS

The number of saved data points should be settable by the user.

3.10 Text Manipulators

These actuators display and read text. *Pnl_typein* gets text from the user, *pnl_typeout* displays and allows selection of multi-line text buffers in a scrolling window. *Pnl_label* displays short text strings in a fixed position in a window.

NAME

pnl_typein

A *pnl_typein* is used to acquire text input from the user. It is supplied with a proto-string argument that appears as the default input. After selecting the typein, the user may delete, or backspace over the default string to replace it with his own typed input.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  char *str;
  int len;
} Typein;
```

str: the string entered, also used as the default string.
len: the maximum length of the entered string.

VALUE

Pnl_typein returns the string entered by the user. This is accessed through the actuator-specific data, not the val field.

NAME

pnl_typeout

Pnl_typeout displays multi-line text buffers in a scrolling window. The user operates an elevator and scroll buttons to position the text in the window. The user may select text by pointing and dragging in the window.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int mode;
  char *buf;
  char *delimstr;
  int start;
  int dot;
  int mark;
  int col,lin;
  int len;
  int size;
  Coord ch, cw, cd;
} Typeout;
```

mode:	typeout modes
buf:	text buffer
delimstr:	characters that serve as delimiters in automatic word select mode.
start:	the index of the first character to display.
dot:	the current selection point.
mark:	the last selection point (where mouse went down).
col, lin:	size of the typeout text window, in columns and lines of text.
len:	number of characters in buffer.
size:	maximum buffer length.

MODES

These modes may be or'ed together.

```
#define PNL_TOM_NORMAL 0x00
#define PNL_TOM_NOCURSOR 0x01
#define PNL_TOM_NOREGION 0x02
```

PNL_TOM_NORMAL: no special modes, cursor and region visible.

PNL_TOM_NOCURSOR: do not draw cursor rectangle at dot.

PNL_TOM_NOREGION: do not highlight text between dot and mark.

VALUE

The **val** field of *pnl_typeout* is unused. *Pnl_typeout* displays a percentage of a text buffer. To change the text that is displayed, the programmer may change the contents of the buffer, or change the value of the START field in the actuator-specific data structure.

APPEARANCE

Pnl_typeout is a light rectangle, with text inside drawn dark. A thin rectangular stripe runs along the right edge. The ends of the stripe are marked with double arrow buttons.

SPECIAL FUNCTIONS

Pnl_typeout provides the *tprint()* (cf) call to append text to the end of the typeout's text buffer and position the buffer in the scrolling window so that the newly added text is visible.

NAME

pnl_label

Pnl_label **is intended for short, single line text display. Text is drawn with no special outline in a dark, contrasting color on the normal panel background.**

VALUE

The val field of *pnl_label* is unused.

APPEARANCE

Pnl_label **text looks like the labels for other actuators.**

ADDING OPERATIONS

none

FIXING OPERATIONS

none

3.11 Mouse

NAME

pnl_mouse

Pnl_mouse **is a special invisible actuator that returns the mouse position and allows the programmer to attach functions to the mouse buttons.**

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  Coord x;
  Coord y;
} Mouse;
```

x: the x coordinate of the mouse in pixels relative to screen origin
y: the y coordinate of the mouse in pixels relative to screen origin

VALUE

Pnl_mouse **returns the position of the mouse in screen coordinates relative to the screen origin when the mouse button is depressed in a user window. Mouse position data will continue to be returned until the mouse button is released even if the mouse is moved out of the user window.** ***Pnl_mouse*** **is not activated when the mouse is clicked in a control panel window.**

APPEARANCE

none

ADDING OPERATIONS

When added, *pnl_mouse* allocates space for its actuator-specific data structure and registers itself with the Panel Library as *pnl_mouse_act* (a Panel Library global variable).

3.12 Compound Actuators

The Panel Library supports compound actuators, or actuators to which sub-actuators have been added. Programmers may build their own hierarchical actuators using this capability, adding functionality by attaching one kind of working actuator under another that has some other desirable property. The grouping actuators are used this way. The Panel Library also provides some compound actuators that have already been assembled and may be operated as a single unit like the simple actuators described above.

3.12.1 Multislider

NAME

pnl_multislider

Pnl_multislider resembles *pnl_slider*, with the ability to add additional slider bars. The bars are manipulated independently and may have their own action functions. When created, pnl_multislider makes a number of sliderbars as specified in the actuator-specific data structure field N, and spaces them evenly over the range of the slider.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int mode;
  int n;
  float finefactor;
  Coord wsave;
  Actuator *sa;
  Coord bh;
  Coord clrx, clry, clrw, clrh;
  void (*acttype)();
} Multislider;
```

mode:	multislider mode
n:	number of sliderbars to create initially. default ???
finefactor:	ratio of value motion to mouse motion when in fine control mode. Default value is PNL_FINE_CONTROL_FACTOR (1/20).
valsave:	temp store for value between calls (internal).
sa:	pointer to the last selected sliderbar.
bh:	sliderbar height.
clrx, clry, clrh, clrw:	a rectangle that bounds the area used by the sliderbars' labels.
acttype:	the actuator init function to use to create sliderbars.

MODES

Modes from these two groups may be or'ed together. Only one of these three modes may used at a time.

```
#define PNL_MSM_FREE 0x00
#define PNL_MSM_ORDERED 0x01
#define PNL_MSM_CONSTRAINED 0x02
```

Only one of these two modes may be used at a time.

```
#define PNL_MSM_ADD 0x04
#define PNL_MSM_DELETE 0x08
```

PNL_MSM_FREE: sliderbars are free to be moved anywhere on the multislider.

PNL_MSM_ORDERED: sliderbars other than that currently being adjusted with the mouse will move to maintain their original ordering.

PNL_MSM_CONSTRAINED: the sliderbar being adjusted will not be allowed to move past its neighbors.

PNL_MSM_ADD: subsequent clicks in the multislider will create and add sliderbars of type acttype to the multislider.

PNL_MSM_DELETE: subsequent clicks in the multislider will delete the nearest sliderbar.

VALUE

Pnl_multislider takes as its current value (**val**) the position of the sliderbar being manipulated. Its actuator-specific data structure field SA is set to point to the Actuator descriptor of the sliderbar. All sliderbars' **extval** fields are set to reflect their position on the multislider.

APPEARANCE

Pnl_multislider is a long narrow rectangle identical in appearance to *Pnl_slider*. The interior of *pnl_multislider* is populated with a variable number of sliderbars whose default height is half that of those inside *pnl_slider*. The sliderbars may have labels, an area for which will be cleared by *pnl_multislider*.

ADDING OPERATIONS

When *pnl_mslider* is added to a panel, it creates a number of slider bars specified as N (default 5) in the actuator-specific data structure.

When subactuators are added to *pnl_multislider*, they take their position on the multislider according to their **extval** field. Sliderbars may not be added in between an end and an already existing sliderbar that is positioned at the end.

FIXING OPERATIONS

When fixed, *pnl_multislider* calculates the smallest rectangle that will cover all possible positions of its subactuators' labels. This area is cleared when the multislider is redrawn.

NAME

pnl_multislider_bar

pnl_multislider_open_bar

Pnl_multislider_bar and *pnl_multislider_open_bar* are very simple actuators designed to serve as sliderbars in *pnl_multislider*. They are basically rectangular buttons that do not change appearance to reflect being selected with the mouse.

VALUE

These actuators do not change their value.

APPEARANCE

These actuators are featureless rectangles. *Pnl_multislider_bar* is darker than *pnl_multislider_open_bar*.

ADDING OPERATIONS

none

FIXING OPERATIONS

When fixed, *pnl_multislider_bar* and *pnl_multislider_open_bar* calculate their y position based on their **extval** fields.

3.12.2 Grouping Actuators

The grouping actuators allow the programmer to treat a number of actuators as a single actuator, and to modify the behavior of exiisting actuators.

NAME

pnl_frame

Pnl_frame is the basic grouping actuator. Frames are used to impose a new viewing transformation on subactuators. Actuators drawn inside a frame may have their own scalefactor and origin. Manipulations applied to the frame are passed on to subactuators, for example, it the frame is made invisible, all subactuators disappear as well, if the frame is moved, its contents move, or if the frame is made unselectable, no actuator inside it may be selected.

Frames are used (instead of panels) to group actuators when you don't want to create a separate IRIS window.

For (scant) information on *pnl_ viewframe*, see New Actuators.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int mode;
  Coord offx, offy;
  Coord minx;
  Coord maxx;
  Coord miny;
  Coord maxy;
} Frame;
```

mode:	frame modes
offx, offy:	origin offset for frame contents
minx, miny, maxx, maxy:	visible bounding box for all subactuators and labels, these dimensions are mapped to the edge of the frame.

MODES

Use only one of these modes at a time.

PNL_FM_FREE: The frame will resize to accommodate its subactuators.

PNL_FM_FIXED: The frame will not resize its boundary nor rescale its subactuators.

PNL_FM_FIXED_SIZE: The frame will rescale its subactuators to fit inside its fixed boundaries.

VALUE

Pnl_frame returns the value of its current actuator, that is, the highest-level subactuator that has returned a value.

APPEARANCE

Pnl_frame is a beveled rectangle whose background color is that of regular panels.

ADDING OPERATIONS

When *pnl_frame* is added to a panel, it fixes itself.

When subactuators are added to *pnl_frame*, they are added to the frame's actuator list (AL) and fixed.

FIXING OPERATIONS

Fixing *pnl_frame* causes it to recalculate x and y limits of its subactuators (MINX, MINY, MAXX, MAXY in its actuator-specific data structure). If the frame is in PNL_FM_FIXED_SIZE mode, its scalefactor is calculated to allow all actuators to be displayed. In PNL_FM_FREE mode, the frame itself may change shape to accommodate any changes in the size or position of its subactuators. In PNL_FM_FIXED mode, neither of these adjustments is performed.

NAME

pnl_icon

Pnl_icon takes a single actuator and adds a momentary iconification behavior to it. That is, the icon appears on a panel as a small rectangle until selected with the mouse, whereupon it changes into its other visible form, to wit, that of its subactuator.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int mode;
  Coord xstowed, ystowed, wstowed, hstowed;
  Coord xopen, yopen, wopen, hopen;
  char *labelsave;
} Icon;
```

mode:	icon modes.
xstowed, ystowed, wstowed, hstowed:	size ond position of the icon when iconified.
xopen, yopen, wopen, hopen:	size ond position of the icon when expanded.
labelsave:	temporary storage for icon's label.

MODES

Use only one of these modes at a time. The newvalfunc for *pnl_icon* sets the icon mode so any programmer use of mode must be re-instated after the newvalfunc is called (for example in activefunc and upfunc).

```
#define PNL_IM_STOWED 0x01
#define PNL_IM_OPEN 0x02
```

PNL_IM_OPEN: icon currently expanded.

PNL_IM_STOWED: icon currently iconified.

VALUE

Pnl_icon returns the value of its subactuator. After opening the icon (by selecting it) its subactuator is active, as though it had been clicked upon. Both the icon and its subactuator stay active until the mouse button is released, whereupon the icon resumes its iconified appearance.

APPEARANCE

Pnl_icon is a small retangle with a label inside it, similar to *pnl_wide_button.*

ADDING OPERATIONS

When *pnl_icon* is added to a panel, its **label** pointer is saved in its **labelsave** actuator-specific data structure field.

Adding a subactuator to *pnl_icon* causes it to be placed on the icon's actuator list **al**, and to become its current actuator **ca**. The subactuator is made invisible **visible=false.**

FIXING OPERATIONS

Fixing pnl_icon saves its current size and position as **xstowed, ystowed, wstowed, hstowed** in its actuator-specific data structure. The size of its open state is taken from the current size of its subactuator (**wopen** and **hopen**) and **xopen** and **yopen** are calculated so that the top of the icon and the expanded subactuator coincide, with the subactuator centered on the over the icon from side to side.

NAME

pnl_cycle

Pnl_cycle displays one of a number of actuators that have been added to it. The user may click buttons to cycle left or right through the list of subactuators.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  int mode;
  Actuator *shiftleftbutton;
  Actuator *shiftrightbutton;
  Actuator *frame;
  Alist *memberlist, *currentmember;
} Cycle;
```

mode:	cycle modes
shiftrightbutton, shiftleftbutton:	button actuators to change subactuator currently displayed.
frame:	frame, sized to accommodate largest subactuator and the two shift buttons.
memberlist:	list of actuators to cycle through.
currentmember:	member currently displayed.

MODES

These modes are used while creating the cycle. The programmer will not likely set the mode explicitly.

```
#define PNL_CM_NORMAL 0x00
#define PNL_CM_UNDER_CONSTRUCTION 0x01
```

PNL_CM_NORMAL: normal mode.

PNL_CM_UNDER_CONSTRUCTION: used while building the actuator.

VALUE

Pnl_cycle returns the value of its current subactuator.

APPEARANCE

Pnl_cycle is a rectangle with two arrow buttons in the upper left. The rest of the rectangle displays the current sub-actuator.

ADDING OPERATIONS

When subactuators are added to *pnl_cycle*, they are added to the cycle's memberlist al and fixed. The most recently added subactuator is the first visible. The frame's size is widened to include the new actuator if necessary. The cycle is fixed.

FIXING OPERATIONS

When *pnl_cycle* is fixed, the buttons are located to the right edge of the frame, and the cycle's size is set to be equal to that of its frame.

NAME

pnl_scroll

Pnl_scroll is a grouping actuator that provides a means for scrolling back and forth over the area of a large group of actuators. Actuators are added to pnl_scroll just as they are to a frame, but the scroll has a fixed size, normally smaller than the frame required to display full-sized versions of the subactuators would be. The scroll displays a portion of the larger "frame" thus created and the user may operate sliders to scroll back and forth to reveal other parts of the control panel.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  Actuator *vslider, *hslider;
  Actuator *frame, *subframe;
} Scroll;
```

vslider,
hslider: the scrolling control sliders.
frame: the frame containing the entire scroll.
subframe: the frame containing the added subactuators.

VALUE

Pnl_scroll returns the value of its current subactuator.

APPEARANCE

Pnl_scroll is a rectangle with long narrow sliders across the bottom and along the right edge. A smaller rectangle abuts these sliders and the top and left edges. Inside the smaller rectangle, the scroll's added subactuators are displayed.

ADDING OPERATIONS

When *pnl_scroll* is added to a panel, it fixes itself.

When subactuators are added to *pnl_scroll*, they are added to the actuator list al of subframe. Subframe's origin offset is set to correspond to the lower left corner of it's bounding rectangle, as are the origin offset control sliders.

FIXING OPERATIONS

The components of the scroll are sized and located within the size given for the scroll actuator itself. The enclosing frame is fixed, and then its bounding box is trimmed to eliminate margin spacing. The sliders' output values are scaled to match the bounding rectangle of the subframe, and finally the subframe's bounding rectangle is set to clip at its edges.

3.12.3 Menus

The actuators listed in this section provide a menu actuator that makes its subactuators active just by dragging the mouse over them, a menu built into *pnl_icon* to form "pop-up" menus, a special kind of menu that pops to the side for use as a sub-menu, and a generic menu entry.

NAME

pnl_menu

Pnl_menu is a grouping actuator that displays its subactuators in a vertical column. After clicking in the menu, actuators that the mouse are dragged over become active. Releasing the mouse button calls the upfunc for the currently active actuator.

ACTUATOR-SPECIFIC DATA

```
typedef struct {
  Coord th;
} Menu;
```

th: title height.

VALUE

Pnl_menu returns an integer corresponding to currently active subactuator.

APPEARANCE

Pnl_menu is a rectangle, wide enough for its widest menu item. The menu items are stacked vertically.

ADDING OPERATIONS

When *pnl_menu* is added to a panel, it widens itself to accommodate its label, if necessary.

When a subactuator is added to *pnl_menu*, it is added to the menu's actuator list **al**, the menu is widened to accommodate the new subactuator if necessary, and the other subactuators are moved to place the new one at the bottom. All the subactuators are fixed.

FIXING OPERATIONS

When fixed, *pnl_menu* fixes its subactuators and recalculates its width.

NAME

pnl_icon_menu

Pnl_icon_menu is *pnl_menu* attached to *pnl_icon*. *Pnl_icon_menu* in this way inherits the behavior of expanding from a small icon to implement a type of pop-up menu.

VALUE

Pnl_icon_menu returns an integer reflecting its active menu choice.

APPEARANCE

Pnl_icon_menu appears identical to *pnl_icon*. When selected, it appears as *pnl_menu*.

ADDING OPERATIONS

When *pnl_icon_menu* is added to a panel, the addfunc for *pnl_icon* is called. *Pnl_icon_menu* then creates a *pnl_menu*, and adds it as a subactuator to itself. It then sets its current actuator ca to be its menu.

Actuators added to *pnl_icon_menu* after its menu has been created are processed by the addsubfunc of *pnl_menu* to be subactuators of the menu.

FIXING OPERATIONS

Fixing*pnl_icon_menu* causes the menu to be fixed (the default behavior of *pnl_icon*).

NAME

pnl_sub_menu

Pnl_sub_menu behaves very similarly to*pnl_icon_menu*. It is intended to be used as a menu item in other menu lists. when selected, it expands to right of its icon, rather than centered over it.

VALUE

Pnl_sub_menu returns an integer reflecting its active menu choice.

APPEARANCE

Pnl_sub_menu appears identical to*pnl_icon_menu*, except that its menu appears to the right of its icon.

ADDING OPERATIONS

When*pnl_sub_menu* is added to a panel, it calls *pnl_icon*'s addfunc, and then creates a *pnl_menu*, which it adds to itself, using the addsubfunc of *pnl_icon*. It then sets its addsubfunc to be its own, so that subsequently added subactuators are processed differently.

Actuators added to *pnl_sub_menu* are added to the menu by *pnl_menu*'s addsubfunc.

FIXING OPERATIONS

When fixed, *pnl_sub_menu* performs the normal*pnl_icon* fixing operations and then offsets the open position to the right by modifying **xopen** in the actuator-specific data structure.

NAME

pnl_menu_item

Pnl_menu_item **is a kind of wide button (see *pnl_wide_button*) whose color is dependent on its selection status.**

VALUE

Pnl_menu_item **returns maxval when active (selected), minval otherwise.**

APPEARANCE

Pnl_menu_item **appears as a wide non-highlighted rectangle with its label centered upon it when val equals minval. When val equals maxval, *pnl_menu_item* becomes highlighted, with its label drawn in inverse.**

ADDING OPERATIONS

none

FIXING OPERATIONS

When fixed *pnl_menu_item* recalculates its width to accommodate a possibly changed label length.

3.12.4 Signal

New info to be added here.

3.13 New Actuators–Viewframe and ??

Actuators are added to the Panel Library as the need for them arises. For the latest information, see the source directory and the latest demos. For example, for details on *pnl_viewframe*, a framing actuator that allows the application program to draw an image within a panel, rather than as a separate window, see viewframe.c on the source tape.

Chapter 4

Structure Fields

4.1 List of Fields

The following list of individual structure fields is ordered pedagogically, and does not reflect the order of appearance of the fields in the actual structure definition.

Panel Fields

```
short id;                       /* unique id */
short gid;                      /* window number of this panels window */
short usergid;                  /* window number of one of the user's windows */

Actuator *a;                    /* current actuator */
Actuator *al;                   /* actuator list */
Actuator *lastgroup;            /* last actuator added to a group */
Alist *autolist; Panel *next;   /* next panel in panel list */

Boolean active;                 /* currently selected */
Boolean enabled;                /* selection allowed */
Boolean visible;                /* make a window for this panel */

long x, y, w, h;                /* screen location of the window and its size */
```

```
Coord minx;                /* bounding box enclosing all actuators and labels */
Coord maxx;
Coord miny;
Coord maxy;

Coord cw,ch;               /* char width and height */
char *label;               /* window title */

Object vobj;               /* viewing transformations */
float ppu;                 /* pixels per unit */

void (*fixfunc)(Actuator *);
void (*drawfunc)(Panel *);

void (*downfunc)(Panel *);
void (*activefunc)(Panel *);
void (*upfunc)(Panel *);

int somedirty;             /* there is a dirty act on this panel */
int dirtycnt;              /* panel needs to be redrawn */
```

Actuator Fields

```
short id;          /* unique id */
int type;          /* type id */

Boolean active;    /* currently selected with mouse */

Panel *p;          /* pointer to enclosing panel */
Actuator *pa;      /* parent actuator */
Actuator *ca;      /* currently active subactuator (if any) */
Actuator *al;      /* list of sub-actuators */
Actuator *group;   /* ring of associated actuators */
Actuator *next;    /* next actuator in p's al */
```

```
int na;             /* number of sub-actuators */

Coord x, y;         /* location */
Coord w, h;         /* size */

float scalefactor;  /* scale of subactuators */
Coord lx, ly, lw, lh;    /* location and size of label */
Coord ld;           /* descender size */
char *label;
int labeltype;      /* location of the label relative to the actuator */

float val;          /* the value */
float minval, maxval;  /* limits for val */
float initval;      /* reset value */
float extval;       /* value in context of parent actuator */

Device key;         /* keyboard equivalent */

void (*initfunc)();
Boolean (*pickfunc)(Actuator *, Panel *, Coord, Coord);
void (*addfunc)();
void (*addsubfunc)(Actuator *, Actuator *);
void (*delfunc)(Actuator *);
void (*newvalfunc)(Actuator *, Panel *, Coord, Coord);
void (*fixfunc)(Actuator *);
void (*drawfunc)(Actuator *);
void (*dumpfunc)(Actuator *, int);
void (*loadfunc)(Actuator *, int);

void (*downfunc)(Actuator *);   /* what I do when moused */
void (*activefunc)(Actuator *); /* what I do while moused */
void (*upfunc)(Actuator *);     /* what I do when released */

char *u;        /* pointer to arbitrary user data */
char *data;     /* pointer to data peculiar to a particular actuator */
int datasize;   /* size of data struct plus everything it points to */
```

```
Boolean automatic;   /* true ==> newvalfunc called every dopanel */
Boolean selectable;  /* false ==> unpickable, newvalfunc never called */
Boolean visible;     /* does this actuator have a visible manifestation? */
Boolean beveled;     /*is this actuator got a beveled edge? */
int dirtycnt;        /* damage control */
```

4.2 Field Descriptions

a

(Panel field)

Actuator *a;

A is a pointer to the Panel's current actuator. It is set in *pnl_dopanel()*. It corresponds to *pnl_ca*. The current actuator is the highest level actuator under the mouse when the mouse button goes down.

active

(Actuator field)

Boolean active;

Active is true when the actuator is currently being moused. It is set in *pnl_dopanel()*. Whenever *pnl_dopanel()* returns an actuator, **active** is set. When the mouse button goes up, the library sets **active** false, and calls the **newvalfunc** for the actuator one more time. This allows the actuator to change itself and any subactuators to their quiescent state.

active

(Panel field)

Boolean active;

Active is true when the mouse has clicked over the panel. There need not be an active actuator in the panel.

activefunc

(Actuator field)

void (*activefunc)(Actuator *);

Activefunc is one of the actuator's action functions. **Activefunc** is called for a selected actuator whenever the mouse button

is down. **Activefunc** is supplied with a pointer to the actuator that is being processed when it is called.

see also

downfunc, upfunc, "Action Functions"

activefunc

(Panel field)

void (*activefunc)(Panel *);

Activefunc is one of the panel's action functions. **Activefunc** is called for a selected panel whenever the mouse button is down. **Activefunc** is supplied with a pointer to the panel that is being processed when it is called.

see also

downfunc, upfunc, "Action Functions"

addfunc

(Actuator field)

void (*addfunc)();

Addfunc is one of the actuator's behavior functions. An actuator's **addfunc** is called by *pnl_addact()* after performing generic initialization of the actuator.

see also

initfunc, pickfunc, addsubfunc, delfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc, "Behavior Functions".

addsubfunc

(Actuator field)

void (*addsubfunc)(Actuator *, Actuator *);

Addsubfunc is one of the actuator's behavior functions. An actuator's **addsubfunc** is called by *pnl_addsubact()* after performing generic initialization of the subactuator.

see also

initfunc, pickfunc, addfunc, addsubfunc, delfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc, "Behavior Functions".

al

(Actuator field)

Actuator *al;

Al is the head pointer of the actuator's [[sub?]]actuator list. Subactuators in **al** are linked through their **next** field. As subactuators are added to most actuators with *pnl_addsubact()*, they are usually added to **al**. Since maintaining **al** is the responsibility of each specific actuator, the exact placement or even the appearance of a subactuator in **al** is not guaranteed. **Al** is used by the library to apply global operations on actuators, therefore it is intended that each actuator appear in an **al** of some actuator of [[or?]] panel.

see also

pnl_addsubact(), **addsubfunc, next.**

al

(Panel field)

Actuator *al;

Al is the head pointer of the panel's actuator list. Actuators are added to the head of a panel's **al** by *pnl_addact()*, therefore **al** is in reverse order of the sequence of *pnl_addact()* calls.

see also

pnl_addact(), *pnl_addsubact()*, **addfunc, next.**

autolist

(Panel field)

Alist *autolist;

Autolist is the head pointer of a list of actuators whose **newvalfuncs** are to be called every time *pnl_dopanel* is called. Actuators are added to **autolist** by *pnl_addact()* and *pnl_addsubact()* if their **automatic** field is set to true. Their **newvalfuncs** will continue to be called unless their **automatic** or **enabled** fields are set to false.

see also

pnl_addact(), *pnl_addsubact()*, **automatic**, **enabled**, **newvalfunc**.

automatic

(Actuator field)

Boolean automatic;

Automatic is a boolean field indicating to *pnl_addact()* and *pnl_addsubact()* whether the actuator is to be added to the enclosing panel's **autolist**. Automatic actuators have their **newvalfuncs** called every time *pnl_dopanel()* is called. The actuator will be treated as automatic while **automatic** and **enabled** are set to true.

see also

pnl_addact(), *pnl_addsubact()*, **automatic**, **enabled**, **newvalfunc**.

beveled

(Actuator field)

Boolean beveled;

Actuators whose **beveled** field is set to true are drawn with a beveled rectangular border.

see also

drawfunc.

ca

(Actuator field)

Actuator *ca;

Ca is an actuator's current subactuator. Although maintenance of the **ca** field is the responsibility of each actuator, **ca** usually provides a pointer to the subactuator currently being manipulated with the mouse, or after an actuator has become inactive, the last subactuator selected. If not pointing directly to the active subactuator, **ca** usually is part of a path to it.

ch

(Panel field)

Coord ch;

Ch is total (including descender) character height in the current worldspace panel coordinates.

see also

cw

cw

(Panel field)

Coord cw;

Cw is the character width in the current worldspace panel coordinates.

data

(Actuator field)

char *data;

Data provides a pointer to an actuator's actuator-specific data structure. Actuators allocate space and assign data to its address in their **initfuncs**. Data must be dereferenced in order to use some actuators. This may be done by using the PNL_ACCESS() macro, or by declaring an auxiliary pointer as in these examples.

examples

accessing actuator-specific data using PNL_ACCESS

```
Actuator *a;

my_x=PNL_ACCESS(Puck, a, x); my_y=PNL_ACCESS(Puck, a, y);
```

accessing actuator-specific data using an auxiliary pointer.

```
Actuator *a;  Puck *ad;

ad=(Puck *)a->data; my_x=ad->x; my_y=ad->y;
```

see also

PNL_ACCESS, **datasize, initfunc.**

datasize

(Actuator field)

int datasize;

Datasize is the size in bytes of the actuator-specific data structure referenced by data. It may be used by **loadfunc** and **dumpfunc** to determine how much data to transfer.

see also

data, loadfunc, dumpfunc.

delfunc

(Actuator field)

void (*delfunc)(Actuator *);

Delfunc is one of the actuator's behavior functions. An actuator's **delfunc** is called by *pnl_delact()* after performing generic deletion processing on the library's data structures.

see also

Pnl_delact(), **initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc,** "Behavior Functions".

dirtycnt

(Actuator field)

int dirtycnt;

The **dirtycnt** actuator field is used to record the number of times an actuator must be drawn to bring it up to date in both the front and back buffers. When an actuator changes its value, it sets the **dirtycnt** to two, indicating that both front and back buffers are out of date. This should be done in **newvalfuncs** with *pnl_setdirty()*, which also sets the the current panel's **somedirty**, to indicate that a traversal of that panel's actuators is required by the drawing function. User code should indicate that a redraw is required by using *pnl_fixact()* which calls *pnl_setdirty()*. As the actuator is drawn, **dirtycnt** is decremented. **Drawfuncs** generally do not redraw the actuator unless **dirtycnt** is greater than zero. Compound actuators generally propagate non-zero **dirtycnts** downward to their subactuators, so that marking a higher-level actuator dirty will cause all its subactuators to be drawn.

see also

Pnl_setdirty(), *pnl_fixact()*, **drawfunc, fixfunc, newvalfunc.**

dirtycnt

(Panel field)

int dirtycnt;

The **dirtycnt** panel field is used to record the number of times a panel must be redrawn to bring it up to date in both the front and back buffers. It is set to two when a panel has been reshaped or moved, and as a result of calling *pnl_fixpanel()*. When a panel

is drawn, **dirtycnt** is decremented. When **dirtycnt** is non-zero, every actuator and their subactuators on the panel are drawn.

see also

pnl_fixpanel(), **drawfunc**, **fixfunc**.

downfunc

(Actuator field)

void (*downfunc)(Actuator *);

Downfunc is one of the actuator's action functions. **Downfunc** is called for a selected actuator when the mouse button goes down. **Downfunc** is supplied with a pointer to the actuator that is being processed when it is called.

see also

activefunc, **upfunc**, "Action Functions"

downfunc

(Panel field)

void (*downfunc)(Panel *);

Downfunc is one of the panel's action functions. **Downfunc** is called for a selected panel when the mouse button goes down. **Downfunc** is supplied with a pointer to the panel that is being processed when it is called.

see also

activefunc, **upfunc**, "Action Functions"

drawfunc

(Actuator field)

void (*drawfunc)(Actuator *);

Drawfunc is one of the actuator's behavior functions. **Drawfunc** does graphics drawing to paint the visual representation of the actuator into the bitplanes.

see also

Pnl_drawact(), **initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc, "Behavior Functions".**

drawfunc

(Panel field)

void (*drawfunc)(Panel *);

Drawfunc is one of the Panel's behavior functions. The default drawfunc for panels calls the drawfuncs of each of its actuators if any of them have been marked dirty with *pnl_setdirty()*.

see also

initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc, "Behavior Functions".

dumpfunc

(Actuator field)

void (*dumpfunc)(Actuator *, int);

Dumpfunc is one of the actuator's behavior functions. Dumpfunc is called when dumping actuator state to a script file. Note: this feature doesn't really work in the current release of the Library.

see also

initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc, "Scripting", "Behavior Functions".

enabled

(Panel field)

Boolean enabled;

Enabled controls whether or not controls on a panel may be manipulated using the mouse. Note: This feature is not implemented, i.e., setting or clearing this field has no effect on Library operation.

extval

(Actuator field)

float extval;

Extval is used to express the value of a subactuator in the context of its parent actuator. For example, actuators that are serving as sliderbars in a multislider have their **extval** set to reflect their position in the multislider. Their **val** fields continue to reflect their own values, for example, **minval** or **maxval** for buttons.

see also

val, minval, maxval, initval.

fixfunc

(Actuator field)

void (*fixfunc)(Actuator *);

Fixfunc is one of the Actuator's behavior functions. **Fixfunc** is called by *pnl_fixact()* to cause the panel to recalculate its size and appearance based on changes that may have been made to its value, label, or visibility of its subactuators.

see also

pnl_fixact(), **initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc,** "Behavior Functions".

fixfunc

(Panel field)

void (*fixfunc)(Actuator *);

Fixfunc is one of the Panel's behavior functions. **Fixfunc** is called by *pnl_fixpanel()* to cause the panel to recalculate its size and appearance based on changes that may have been made to the position or visibility of its actuators.

see also

pnl_fixpanel(), **initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc,** "Behavior Functions".

gid

(Panel field)

short gid;

Gid is the graphics **id** of the window manager graphport that the panel is drawn in.

group

(Actuator field)

Actuator *group;

The **group** pointer is used to implement a ring containing actuators that are in a single group. Actuators are grouped to allow them to alter one another's value when any one of them in activated. For example, (the only example, currently) radio buttons use the group ring to unset all other members of the group when any one of them is set. *Pnl_addtogroup()* is used by the Library to add actuators to the currently open group,*pnl_endgroup()* closes a group; grouped actuators subsequently added to the panel after a call to *pnl_endgroup()* will be added to a new group.

see also

pnl_radio_button, pnl_addtogroup(), pnl_endgroup(), lastgroup.

h

(Actuator field)

Coord h;

H is the height, in panel world coordinates, of the actuator. Any bevel appearing around the actuator is not included in **h**. **H** may be changed at any time to change the height of the actuator. Use *pnl_fixact()* after changing **h**.

see also

pnl_fixact(), **w**, **x**, **y**.

h

(Panel field)

long h;

H is the height, in pixels, of the window the panel is drawn in. **H** includes pixels used to draw the window border and titlebar.

see also

w, **x**, **y**.

id

(Actuator field)

short id;

Id is a unique integer assigned to Actuators and Panels at *pnl_addact()* or *pnl_mkpanel()* time. **Id** is used as an index into *pnl_table* to determine the address of Panel and Actuator structures. This is done when playing back a previously recorded script.

see also

pnl_mkpanel().

id

(Panel field)

short id;

Id is a unique integer assigned to Actuators and Panels at *pnl_addact()* or *pnl_mkpanel()* time. **Id** is used as an index into *pnl_table* to determine the address of Panel and Actuator structures. This is done when playing back a previously recorded script.

see also

pnl_addact().

initfunc

(Actuator field)

void (*initfunc)();

Initfunc is one of the Actuator's behavior functions. **Initfunc** is called by *pnl_mkact()* after creating space for and performing generic initialization of the actuator structure. Since **initfunc** is passed to *pnl_mkact()* as a parameter, it is possible to add new types of actuators without making any changes to the Library.

see also

Pnl_mkact(), **initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc,** "Behavior Functions".

initval

(Actuator field)

float initval;

Initval is set by *pnl_addact()* to the initial value of **val**. It may be used later to reset the actuator to its initial state. The default initial value for most actuators is **minval**.

see also

pnl_addact(),**val, extval, minval, maxval.**

key

(Actuator field)

Device key;

Key is the identifier of the key equivalent for the actuator, if any. *Pnl_addact()* queues the device indicated by **key**. Depressing the key is the same as clicking the actuator with the mouse.

label

(Actuator field)

char *label;

Label is a string that appears near or atop the actuator.

see also

labeltype, fixfunc, *pnl_fixact().*

label

(Panel field)

char *label;

Label is a string that appears in the titlebar of the window in which the panel is drawn.

labeltype

(Actuator field)

int labeltype;

Labeltype indicates where, relative to the actuator, the actuator's label should appear. **Labeltype** takes one of the following values:

PNL_LABEL_RIGHT

PNL_LABEL_RIGHT_TOP

PNL_LABEL_UPPER_RIGHT

PNL_LABEL_TOP_RIGHT

PNL_LABEL_TOP
PNL_LABEL_TOP_LEFT
PNL_LABEL_UPPER_LEFT
PNL_LABEL_LEFT_TOP
PNL_LABEL_LEFT
PNL_LABEL_LEFT_BOTTOM
PNL_LABEL_LOWER_LEFT
PNL_LABEL_BOTTOM_LEFT
PNL_LABEL_BOTTOM
PNL_LABEL_BOTTOM_RIGHT
PNL_LABEL_LOWER_RIGHT
PNL_LABEL_RIGHT_BOTTOM
PNL_LABEL_CENTER
PNL_LABEL_NORMAL

PNL_LABEL_RIGHT centers the label to the right of the actuator. The other positions follow in an anti-clockwise rotation from that point. PNL_CENTER centers the label on the actuator, and PNL_NORMAL provides no offset, positioning the label string origin at the actuator's **x** and **y**.

see also

x, y, label.

lastgroup

(Panel field)

Actuator *lastgroup;

Lastgroup stores the address of the actuator last added to an actuator group ring. Actuators subsequently added to this group are inserted into the ring at **lastgroup->group**. *Pnl_endgroup()* sets lastgroup to NULL.

see also

pnl_addtogroup(), lastgroup(), **group.**

ld

(Actuator field)

Coord ld;

Ld is the size in actuator world coordinates of the actuator's label's font's descenders (the parts of the letters that hang below the baseline of the word). It is recalculated by *pnl_fixact()*.

see also

lh, lw, lx, ly

lh

(Actuator field)

Coord lh;

Lh is the size in actuator world coordinates of the actuator's label's overall height. It is recalculated by *pnl_fixact()*.

see also

ld, lw, lx, ly

loadfunc

(Actuator field)

void (*loadfunc)(Actuator *, int);

Loadfunc is one of the actuator's behavior functions. **Loadfunc** is called when loading actuator state from a script file. Note: this feature doesn't really work in the current release of the Library.

see also

initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc, "Scripting", "Behavior Functions"

lw

(Actuator field)

Coord lw;

Lx is the size in actuator world coordinates of the actuator's label's overall width. It is recalculated by *pnl_fixact()*.

see also

ld, lh, lx, ly

lx

(Actuator field)

Coord lx;

Lx is the value in actuator world coordinates of the actuator's label **x** position. It is recalculated by *pnl_fixact()*.

see also

ld, lw, lh, ly

ly

(Actuator field)

Coord ly;

Ly is the value in actuator world coordinates of the actuator's label **y** position. It is recalculated by *pnl_fixact()*.

see also

ld, lw, lx, lh

maxval

(Actuator field)

float maxval;

Maxval is the maximum value an actuator's **val** field may take. For continuous valuators like sliders, **maxval** is used with **minval** to interpolate the value of **val**. For discrete controlers like buttons, **val** is set to **maxval** when the actuator is selected.

see also

val, minval, initval, extval

maxx

(Panel field)

Coord maxx;

Maxx is the maximum extent in the x direction of any actuator (including its label) on the panel. **Maxx** is calculated by *pnl_fixpanel()* and takes account only of visible actuators.

see also

pnl_fixpanel(), **maxy, minx, miny, visible**

maxy

(Panel field)

Coord maxy;

Maxx is the maximum extent in the y direction of any actuator (including its label) on the panel. Maxy is calculated by *pnl_fixpanel()* and takes account only of visible actuators.

see also

pnl_fixpanel(), **maxx, minx, miny, visible**

minval

(Actuator field)

float minval;

Minval is the minimum value an actuator's **val** field may take. For continuous valuators like sliders, **minval** is used with **maxval** to interpolate the value of **val**. For discrete controlers like buttons, **val** is set to **minval** when the actuator is not selected.

see also

val, maxval, initval, extval

minx

(Panel field)

Coord minx;

Minx is the minimum extent in the **x** direction of any actuator (including its label) on the panel. **Minx** is calculated by *pnl_fixpanel()* and takes account only of visible actuators.

see also

pnl_fixpanel(), **maxy, maxx, miny, visible**

miny

(Panel field)

Coord miny;

Miny is the minimum extent in the y direction of any actuator (including its label) on the panel. **Miny** is calculated by *pnl_fixpanel()* and takes account only of visible actuators.

see also

pnl_fixpanel(), **maxy, maxx, minx, visible**

na

(Actuator field)

int na;

Na is the number of the actuator's subactuators. It is incremented by *pnl_addsubact()*.

see also

pnl_addsubact()

newvalfunc

(Actuator field)

void (*newvalfunc)(Actuator *, Panel *, Coord, Coord);

Newvalfunc is one of the actuator's behavior functions. **Newvalfunc** is called via *pnl_newvalact()* when an actuator is first selected with the mouse, and is called repeatedly while the mouse button is depressed to calculate the value of **val** based on the mouse position. When the mouse button is released, **newvalfunc** is called once again to allow the actuator to return to its quiescent state.

see also

Pnl_newvalact(), **val, initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc,** "Behavior Functions"

next

(Actuator field)

Actuator *next;

Next provides the link for the Panel's list of actuators, **al**, and the actuator's list of subactuators, also called **al**. Actuators are added to these lists by *pnl_addact()* when being added to a panel, and by the **addsubfuncs** of their parent actuator when being added to an actuator. This latter behavior allows actuators to manage the order or appearance of subactuators in the **al** depending on the subactuator's role in the actuator.

see also

pnl_addact(), *pnl_addsubact()*, **al**

next

(Panel field)

Panel *next;

Next provides the link for the Library's list of panels, *pnl_pl*. Panels are added at the head of the list by *pnl_mkpanel()*.

see also

pnl_mkpanel(), **al**

p

(Actuator field)

Panel *p;

P points to the Panel the actuator is a member of. An actuator and all its subactuators all have a pointer to the same panel.

pa

(Actuator field)

Actuator *pa;

Pa points to the actuator's parent actuator. **Pa** is NULL if the actuator is not a subactuator. **Pa** is set by *pnl_addsubact()*.

see also

pnl_addsubact()

pickfunc

(Actuator field)

Boolean (*pickfunc)(Actuator *, Panel *, Coord, Coord);

Pickfunc is one of the actuator's behavior functions. **Pickfunc** is called by the Library and some compound actuators like *pnl_frame* to determine whether the mouse lies within the boundaries of the actuator.

see also

initfunc, pickfunc, addfunc, addsubfunc, newvalfunc, fixfunc, drawfunc, dumpfunc, loadfunc, "Behavior Functions"

ppu

(Panel field)

float ppu;

Ppu is the current number of pixels per unit in panel world coordinates. It is used to calculate the size of pixel oriented dimensions (like fonts) in worldspace. It is recalculated when panels are reshaped. Setting **ppu** and calling *pnl_fixpanel()* will resize the panel.

see also

lx, ly, lw, lh, ld, *pnl_fixpanel()*.

scalefactor

(Actuator field)

float scalefactor;

Scalefactor is a global scaling factor that is applied when drawing the actuator and all its subactuators. Note: **scalefactor** is known to be completely implemented only for *pnl_frame* actuators in the current release.

see also

pnl_frame

selectable

(Actuator field)

Boolean selectable;

Selectable allows the actuator to be selected with the mouse. Setting **selectable** FALSE causes it to be drawn with a cross-hatched overlay. Actuators with **selectable** set FALSE are still operated from scripts.

somedirty

(Panel field)

int somedirty;

Somedirty is set by *pnl_setdirty()* to indicate that there is an actuator which needs to be redrawn somewhere on the Panel.

see also

pnl_setdirty(), **dirtycnt**

type

(Actuator field)

int type;

Type is an identfier indicating what type of actuator the actuator is. It is set to one of a number of manifest constants defined in panel.h

u

(Actuator field) char *u;

U is a pointer provided for the user. It is used to attach an arbitrary structure to an actuator. For simple cases using appropriate casts, it can hold the actual data.

upfunc

(Actuator field)

void (*upfunc)(Actuator *);

Upfunc is one of the Actuator's action functions. **Upfunc** is called for a selected actuator when the mouse button goes up. **Upfunc** is supplied with a pointer to the panel that is being processed when it is called.

see also

activefunc, downfunc, "Action Functions"

upfunc

(Panel field)

void (*upfunc)(Panel *);

Upfunc is one of the panel's action functions. **Upfunc** is called for a selected panel when the mouse button goes up. **Upfunc** is supplied with a pointer to the panel that is being processed when it is called.

see also

activefunc, downfunc, "Action Functions"

usergid

(Panel field)

short usergid;

Usergid is the graphics id of the graphport that was the current window when the user called *pnl_mkpanel()*.

see also

pnl_mkpanel()

val

(Actuator field)

float val;

Val is the curent value of the actuator. It is set by the actuator's **newvalfunc**. For continuous valuators like sliders, **val** usually interpolates **minval** and **maxval** based on the position of the mouse within the actuator. For discrete controllers, like buttons, **val** is set to **maxval** when the actuator is selected, and **minval** when it is not. The programmer can set **val** to change the appearance of most actuators. For the change to be visible, *pnl_fixact()* should be called after setting **val**.

see also

minval, maxval, initval, extval, newvalfunc, *pnl_fixact()*

visible

(Actuator field)

Boolean visible;

Visible controls whether or not the Actuator is drawn on the panel it is a member of. Invisible actuators do not contribute to calculations of the panel's extent, therefore, panels will change size to reflect changes in the visibility of its actuators. Use *pnl_fixpanel()* after changing an actuator's visibility.

see also

pnl_fixpanel()

visible

(Panel field)

Boolean visible;

Visible controls whether or not the Panel is drawn. Setting **visible** to FALSE will delete the window manager window for the Panel. Making an invisible window visible will cause the Library to create a window and draw the panel in it. When windows are created, they are 'in front of' other windows on the screen. Use *pnl_fixpanel()* after changing an actuator's visibility.

vobj

(Panel field)

Object vobj;

Vobj is a GL graphics object containing the viewing transformation for the Panel. It is used by the Library to map mouse screen location into panel world coordinates.

w

(Actuator field)

Coord w;

W is the width, in panel world coordinates, of the actuator. Any bevel appearing around the actuator is not included in **w**. **W** may be changed at any time to change the width of the actuator. Use *pnl_fixact()* after changing **w**.

see also

pnl_fixact(), **h**, **x**, **y**

w

(Panel field)

long w;

W is the width in pixels of the window the Panel is drawn in. **W** includes pixels used to draw the window's borders.

see also

h, **x**, **y**

y

(Actuator field)

Coord y;

Y is the y position, in panel world coordinates, of the actuator within its panel or parent actuator. **Y** may be changed at any time to change the position of the actuator. Use *pnl_fixact()* after changing **y**.

see also

pnl_fixact(),**h**, **w**, **x**

y

(Panel field)

long y;

Y is the y screen position in pixels of the window the Panel is drawn in. Y is offset to account for pixels used to draw the window's borders, i.e., y is the location of the outer edge of the border.

see also

h, w, x

Chapter 5

Global Variables

The Panel Library uses a number of global variables to maintain and communicate its internal state. The programmer may read these variables to gain insight into the current mode and operation of the library, and by setting some of them, may influence the behavior of the library.

5.1 Colors

pnl_white_color
Colorindex pnl_white_color;
initial value: 0

pnl_bevel_light_color
Colorindex pnl_bevel_light_color;
initial value: 0

pnl_normal_color
Colorindex pnl_normal_color;
initial value: 0

pnl_background_color
Colorindex pnl_background_color;
initial value: 0

pnl_other_color
Colorindex pnl_other_color;
initial value: 0

pnl_highlight_color
Colorindex pnl_highlight_color;
initial value: 0

pnl_bevel_dark_color
Colorindex pnl_bevel_dark_color;
initial value: 0

pnl_label_color
Colorindex pnl_label_color;
initial value: 0

pnl_black_color
Colorindex pnl_black_color;
initial value: 0

The Panel Library uses nine locations in the color map to store the RGB definitions of the colors it makes available for drawing actuators. The indices used depend on the number of available bit planes, and are intended to make minimum impact on the colors defined by the SGI utility *makemap(1)*. The indices to be used are stored in the following nine global variables the first time *pnl_dopanel()* is called. To specify different indices, just set them, and if desired, also set the RGB contents of those indices. If the indices have been set to something other than zero, the library will not touch them. This means that you can't use color zero (BLACK) as a Panel Library color unless you set it after the first call to *pnl_dopanel()*.

Pnl_white_color is the brightest color, used for text and markings in highlighted regions. *Pnl_bevel_light_color* is the very light color used for upper-left-hand sides of bevels. *Pnl_normal_color* is a light color used to draw actuators. Regions drawn with *pnl_normal_color* are generally mouse-sensitive. *Pnl_background_color* is a medium tone used to draw the background of panels.

Pnl_other_color provides a darker contrast to *pnl_normal_color*, and is used to draw backgrounds and other parts of actuators. *Pnl_highlight_color* is a dark, saturated color used to draw actuators in their alternate state. *Pnl_highlight_color* should be used sparingly for high-visibility control panel elements. *Pnl_bevel_dark_color* is the very dark color used for the lower-right-hand sides of bevels. *Pnl_label_color* is nearly black, and is intended to give readable contrast for text on all lighter colors. *Pnl_black_color* is essentially black, and used for outlines and other markings.

5.2 Fill Pattern

pnl_fade_pattern_index
short pnl_fade_pattern_index;
initial value: PNL_FADE_PATTERN_INDEX

pnl_fade_pattern_size
short pnl_fade_pattern_size;
initial value: PNL_FADE_PATTERN_SIZE

pnl_fade_pattern
short pnl_fade_pattern[];
initial value: PNL_FADE_PATTERN

The library draws over unselectable actuators with a filled rectangle drawn using a fill pattern. *Pnl_fade_pattern_index* is the index of the hardware fill pattern register to be used. Change *pnl_fade_pattern_index* before calling *pnl_dopanel()* to use a different register. *Pnl_fade_pattern_size* is the fill pattern size selector value for the library's fade pattern. *Pnl_fade_pattern* is the array definition of the pattern.

5.3 Panel and Actuator Structures

pnl_table
char *pnl_table[PNL_TABLE_SIZE];
initial values: NULL

pnl_id
int pnl_id;
initial value: 0

Pnl_table holds pointers to all Panels and Actuators. It is used primarily to find the current location of a structure when reading scripts. *Pnl_id* is the index into the *pnl_table*. Every time a panel is created using *pnl_mkact()*, or an actuator is added to a panel using *pnl_addact()*, the slot in *pnl_table* indicated by *pnl_id* is used to store the pointer to the new descriptor, and then *pnl_id* is incremented.

pnl_pl
Panel *pnl_pl;
initial value: NULL

Pnl_pl is a pointer to the head of the library's "panel list". Panels are linked in this list via their **next** field. All panels appear in this list. The end of the list is signified by a NULL**next** field.

pnl_kl
Alist *pnl_kl;
initial value: NULL

Pnl_kl is a pointer to the first list cell in the library's list of actuators with key equivalents. An Alist cell consists of two pointers, **data**, and **next**, in the spirit of a traditional lisp cons cell. A list of cells is connected via their**next** pointers, while their contents are reached via the individual **data** pointers.

pnl_cp
Panel *pnl_cp;
initial value: NULL

Pnl_cp indicates the current (active) panel. It points to the Panel structure describing the panel the mouse was in when the mouse button was clicked. When reading from a script, it points to the panel containing the active actuator as specified by the script (if any). At other times, *pnl_cp* is NULL.

see also

pnl_ca

pnl_ca
Actuator *pnl_ca;
initial value: NULL

Pnl_ca indicates the current (active) actuator. It points to the Actuator structure describing the highest level actuator the mouse was in when the mouse button was clicked. That is, when clicking a compound actuator, *pnl_ca* holds the address of the compound actuator's descriptor, not that of any on the compound actuator's subactuators. When reading from a script, *pnl_ca* points to the active actuator as specified by the script (if any). At other times, *pnl_ca* is NULL. *Pnl_ca* is the value that *pnl_dopanel()* returns.

Note: *Pnl_ca* can be NULL while *pnl_cp* is not if the mouse has hit a panel background.

Note: Future versions of the Panel Library may set *pnl_ca* to the lowest level actuator upon which the mouse is clicked, rather than the highest.

pnl_cp_save

Panel *pnl_cp_save;
initial value: NULL

pnl_ca_save
Actuator *pnl_ca_save;
initial value: NULL

pnl_ca_active_save
Boolean pnl_ca_active_save;
initial value: FALSE

pnl_cp_active_save
Boolean pnl_cp_active_save;
initial value: FALSE

These variables are used to temporarily retain the library's state when reading from a script. *nl_cp_save* and *pnl_ca_save* save the current panel and current actuator, while *pnl_ca_active_save* and *pnl_cp_active_save* save the state of their ACTIVE field.

pnl_mouse_act
Actuator *pnl_mouse_act;
initial value: NULL

Pnl_mouse_act contains a pointer to the mouse actuator, if one has been created by the application programmer. *Pnl_mouse_act* is set by creating a mouse actuator with *pnl_mkact(pnl_mouse)*, and adding the returned actuator to a panel.

5.4 Positions and Dimensions

pnl_ox;
Screencoord pnl_ox;

initial value: none defined

pnl_oy;
Screencoord pnl_oy;
initial value: none defined

> *Pnl_ox* and *pnl_oy* store the origin of the last panel to be created. They are used when the library automatically positions a subsequently added panel. The library will attempt to place the upper-left corner of newly added panel just below the screen position specified by *pnl_ox* and *pnl_oy*.

pnl_mx;
Screencoord pnl_mx;
initial value: none defined

pnl_my;
Screencoord pnl_my;
initial value: none defined

> *Pnl_mx* and *pnl_my* contain the mouse position in absolute screen coordinates. These variables are updated while the mouse button is down, and once more when the button goes up.

pnl_x;
Coord pnl_x;
initial value: none defined

pnl_y;
Coord pnl_y;
initial value: none defined

> *Pnl_x* and *pnl_y* contain the mouse position within the current panel in panel world coordinates. If the mouse button went down in an application's data window, *pnl_x* and *pnl_y* give the mouse

position in absolute screen coordinates. These variables are updated while the mouse button is down, and once more when the button goes up.

Note: *Pnl_x*, and *pnl_y* should probably give the mouse position relative to the origin of the applications data window's when not in a panel.

pnl_char_threshold
float pnl_char_threshold;
initial value: PNL_CHAR_THRESHOLD

Pnl_char_threshold indicates the lowest value for any panel's PPU field for which its text will be draw as text characters. Panels that have been resized so that their PPU is smaller the *pnl_char_threshold* will have their text fields drawn as filled rectangles colored with *pnl_other_color*.

NOTE: Someday the library will support scalable text, and this disgusting hack will no longer be required.

5.5 Scripting

pnl_readscript
Boolean pnl_readscript;
initial value: FALSE

pnl_writescript
Boolean pnl_writescript;
initial value: FALSE

pnl_scriptinfd
int pnl_scriptinfd;

initial value: 0

pnl_scriptoutfd
int pnl_scriptoutfd;
initial value: 0

pnl_scriptinfile
char *pnl_scriptinfile;
initial value: "PANEL.SCRIPT"

pnl_scriptoutfile
char *pnl_scriptoutfile;
initial value: "PANEL.SCRIPT"

Pnl_readscript is true when the library is reading a script, and is false otherwise. *Pnl_writescript* is true when the library is writing a script, and is false otherwise. *Pnl_scriptinfd* and the *pnl_scriptoutfd* are the files descriptors of the input and output scriptfiles. If no file is open for reading or writing, one or the other of these variables will be zero. *Pnl_scriptinfile* and *pnl_scriptoutfile* are copies of strings forming pathnames for successfully opened script files.

int pnl_frame_number
long int pnl_frame_number;
initial value: 0

pnl_delay
int pnl_delay
initial value: 0

pnl_ignore_delay
Boolean pnl_ignore_delay
initial value: FALSE

Pnl_frame_number counts the calls to *pnl_dopanel()*, or frames, that have occurred since the last time the mouse button when

up. When writing a script file, a delay token and the value of *pnl_frame_number* is written to the file when the mouse button goes down. When the library is reading a script, encountering a delay token sets *pnl_delay* to the stored value. The library then suspends reading until *pnl_frame_number* EXCEEDS *pnl_delay*. This has the effect of keeping the real-time playback behavior of the application roughly the same as when recorded. Setting *pnl_ignore_delay* disables suspension of script playback.

NOTE: Clearing *pnl_ignore_delay* shortly after a delay token has been encountered can cause an un-natural pause in the next mouse gesture. To avoid this use logic similar to that of setignoredelay() in script.c.

pnl_action_source
int pnl_action_source;
initial value: PNL_SRC_QUEUE

Pnl_action_source is used to control the caching of mouse events so that the library read from script and live mouse input simultaneously. *Pnl_action_source* takes the value of *PNL_SRC_QUEUE* or *PNL_SRC_SCRIPT*.

pnl_delayvirgin
Boolean pnl_delayvirgin;
initial value: TRUE

This flag is set when beginning to write a script to avoid writing an initial delay token to the script file.

5.6 Library State

pnl_virgin

Boolean pnl_virgin;
initial value: TRUE

Pnl_virgin is true until the first time *pnl_dopanel()* is called. Modifying the value of this variable in not recommended.

pnl_saveuserredraw
Boolean pnl_saveuserredraw;
initial value: FALSE

Pnl_saveuserredraw controls the processing of REDRAW events intended for one of the user's graphics windows. If false, user REDRAW events are simply consumed by the library. If true, the events are requeued so that the user has a chance to check for them with the *pnl_userredraw()* function.

pnl_winsave;
int pnl_winsave;
initial value: none defined

Pnl_winsave saves the gid of the currently active user window when *pnl_dopanel()* is called. Just before *pnl_dopanel()* returns, the window specified by *pnl_winsave* is once again made active.

pnl_justdown;
Boolean pnl_justdown;
initial value: none defined

pnl_justup;
Boolean pnl_justup;
initial value: none defined

pnl_mousedown;

Boolean pnl_mousedown;
initial value: none defined

These three variables reflect the current mouse button state. They are updated once per call to *pnl_dopanel()*. They are stored in script files and restored on playback. *Pnl_justdown* and *pnl_justup* are true for one call to *pnl_dopanel()* when the button first changes state. *Pnl_mousedown* is true when the mouse button is down.

pnl_funcmode
int pnl_funcmode
initial value: PNL_FCNM_NONE

Pnl_funcmode is set to reflect the context in which an action function is called. it allows action functions to modify their behavior based on whether they are being called as a down- , active-, or up-func. *Pnl_funcmode* takes the value *PNL_FCNM_NONE*, *PNL_FCNM_DOWN*, *PNL_FCNM_ACTIVE*, or *PNL_FCNM_UP*.

pnl_shiftkey
Boolean pnl_shiftkey
initial value: FALSE

pnl_controlkey
Boolean pnl_controlkey;
initial value: FALSE

Pnl_shiftkey and *pnl_controlkey* reflect the state of the shift and control keys on the keyboard. They are saved to and restored from script files.

pnl_dont_draw

Boolean pnl_dont_draw;
initial value: FALSE

Setting *pnl_dont_draw* to true disables graphical updating of all panels. This is intended to be used to temporarily avoid the overhead of drawing and displaying control panels in applications where maximum graphical performance is required.

NOTE: The main performance penalty associated with the Panel Library is waiting for panel windows to return from a call to swapbuffers. Since only one window from each process can swap on a given vertical retrace, the maximum frame rate for a Panel Library application is 60/(N+1), where N is the number of panels with dirty actuators on them. To maximize frame rate, minimize N by animating as few actuators as possible, and by setting the VISIBLE field of unneeded panels to false.

pnl_beveled
Boolean pnl_beveled;
initial value: TRUE

Pnl_beveled provides global control over the generation of beveled edges. Setting *pnl_beveled* to false inhibits bevels, providing compatibility with earlier relases of the library.

pnl_dopanel_return_mode
int pnl_dopanel_return_mode;
initial value: PNL_DRM_RETURN_PNL_CA

Pnl_dopanel_return_mode controls the value returned by *pnl_dopanel*. Usually *pnl_dopanel* returns the address of the current actuator when an actuator is being moused, and NULL otherwise. Applications wishing to deactivate the normal functioning of the library may wish to set *pnl_dopanel_return_mode* to *PNL_DRM_RETURN_NULL*. In this state the *pnl_dopanel* always returns NULL.

pnl_panel_bell
Boolean pnl_panel_bell;
initial value: TRUE

Pnl_panel_bell controls whether the keyboard bell is rung when the user mouses a panel with selectable set to FALSE. *Pnl_panel_bell* being true implies that the bell will be rung.

Chapter 6

Behavior Functions

initfunc, pickfunc, newvalfunc, addfunc, addsubfunc, fixfunc, drawfunc, dumpfunc, loadfunc, delfunc.

In addition to the actionfuncs discussed in 1.5.2, the panel library uses pointers to functions to control most of the behavior that make actuators different from each other. In most cases, the default functions will be just fine, and this section can be skipped for the novice user. If you think that you might need to change any of these function pointers, see if you can use actionfuncs instead. If not, consider saving the original value of the function pointer, and calling that from within the function you write. Only in extreme cases should it be necessary to provide a totally new function.
A design concept of the Panel Library is the use of indirect functions to provide behavior that is specific to a particular type of actuator, while letting static code in the library perform generic functions. For example, when *pnl_addact()* is called, generic modification of the Panel Library data structures containing actuators is performed, followed by execution of the actuator's **addfunc**.

The following lists the behavior functions, and gives a brief description of their use and responsibilities. This information is intended to guide the programmer in the development of new actuators.

initfunc

> **Initfunc** takes an Actuator structure that has had its fields initialized to standard values, and customizes it to reflect the properties of a particular type of actuator. The standard initfuncs

are declared globally and are used by application developers as the argument to *pnl_mkact()* to specify which type of actuator to instantiate.

RESPONSIBILITIES

Initfunc is responsible to set the **type**, **w**, and **h** fields of the actuator.

Initfunc sets the value of the behavior function fields to point to functions to implement the behaviors. Behavior functions fields left unassigned are ignored by the library.

The **initfunc** allocates space for actuator-specific data structures and initializes the fields in it. **Data** holds the pointer to this space, and **datasize** its size.

pickfunc

The **pickfunc** examines the mouse position within the panel or parent actuator and determines whether the mouse currently "selects" the actuator. Most actuators use the *_hitact()* function provided by the library which simply compares the mouse position against the actuator's bounding rectangle.

newvalfunc

Each actuator is expected to provide a function for determining the value of its **val** field. This function is given the address of the actuator being processed, the active panel, and the mouse **x** and **y** values. By using this information and the values of global variables describing mouse state, the **newvalfunc** determines its actuator's new value.

RESPONSIBILITIES

Newvalfunc is responsible to behave nicely if called with **active** false. This is to give actuators a chance to change their value when the mouse has been released.

IN COMPOUND ACTUATORS:

Newvalfunc usually calculates x and y coordinates relative to its own origin to determine which subactuator is selected and to propagate to its subactuators' **newvalfuncs**.

If the actuator is not active, any current subactuator is set inactive and its it newvalfunc is called. **Newvalfunc** usually returns with no further processing at this point.

The actuator then determines which subactuator, if any, is active. The subactuator's **selectable** and **visible** fields are usually required to be TRUE, and the subactuator's **pickfunc** may be used. The actuator's **ca** field is usually set to the active subactuator.

If a subactuator is active, its **newvalfunc** is called, using the internal x and y coordinates mentioned above.

Some **newvalfuncs** calculate the subactuator's **extval**, or value in the context of the enclosing actuator, at this time.

The value for the actuator itself is determined, usually duplicating or otherwise derived from the active subactuator's value, and is stored in the actuator's **val** field.

addfunc

Addfunc is called when the actuator is added to a panel or to another actuator as a subactuator. **Addfunc** usually performs further initialization that could not be done by **initfunc** because it requires knowledge of values assigned to the actuator's fields by the application developer. For example, **addfunc** for multislider looks at its actuator-specific **n** field to determine how many sliderbars to add to itself.

addsubfunc

Addsubfunc is called when a subactuator is added to the actuator, it is therefore only used in compound actuators. Its purpose is to link the new subactuator into the actuator structure properly, and update any actuator internal state to accommodate the new subactuator.

RESPONSIBILITIES

If the subactuator is going to be a direct child of the actuator, **addsubfunc** is expected to increment the actuator's **na**, or number of actuator field.

Addsubfunc is expected to set the subactuator's **pa**, or parent actuator field to something reasonable, usually the actuator.

If the subactuator is going to be a direct child of the actuator, **addsubfunc** inserts the subactuator into the actuator's actuator list (**al**). Usually this is done at the head of the list, but some actuators enforce a specific order in the actuator list, and insertion may be elsewhere.

Addsubfunc then updates the subactuator if necessary to reflect its role in the actuator; for example, menu's **addsubfunc** sets the **y** position of the subactuator to reflect the order in which it was added to the actuator.

It is possible that the new subactuator will have implications for previously added subactuators, and addsubfunc may modify them also.

fixfunc

Fixfunc re-calculates an actuator's internal data state values to incorporate any changes that have been made by the programmer.

For example, the **fixfunc** for wide buttons recalculates the width of the button to accommodate any change made in the length of the label string. *Pnl_fixact()*, which calls **fixfunc**, calls itself recursively on each of the actuator's subactuators before calling the actuator's **fixfunc**. Therefore, a **fixfunc** may reliably expect that its subactuators have self-consistent internal data state.

drawfunc

Drawfunc calls IRIS graphics library drawing routines to paint a visible representation of the actuator into the bitplanes. In most cases the appearance of the actuator is a function of its value, and occasionally, the mouse state.

Chapter 7

Sequence of Calls to Functions

The action functions—**downfunc, activefunc, and upfunc**—discussed in Subsection 1.5.2 are called after execution of the **newvalfunc** (behavior function, Chapter 6) for the actuator and its subactuators. Panels may have action functions associated with them also.

Assuming that an actuator A has been added to a panel P, and has in turn had a subactuator SA added to it, and SA, A, and P have a full complement (all three) of action functions, the following indicates in what order the functions are called.

```
event              action
==========================
mouse down
pnl_dopanel()
                   A->newvalfunc(A)
                   SA->newvalfunc(SA)
                   SA->downfunc(SA)
                   SA->activefunc(SA)
                   A->downfunc(A)
                   A->activefunc(A)
                   P->downfunc(P)
                   P->activefunc(P)
```

```
pnl_dopanel()
                A->newvalfunc(A)
                SA->newvalfunc(SA)
                SA->activefunc(SA)
                A->activefunc(A)
                P->activefunc(P)
            .
            .
            .

        continues while mouse button is down
            .
            .
            .

pnl_dopanel()
                A->newvalfunc(A)
                SA->newvalfunc(SA)
                SA->activefunc(SA)
                A->activefunc(A)
                P->activefunc(P)
mouse up
pnl_dopanel()
                A->newvalfunc(A)
                SA->newvalfunc(SA)
                SA->upfunc(SA)
                A->upfunc(A)
                P->upfunc(P)
```

Appendix A. dviiris

This is the help file for dviiris. This unsupported utility is used to display dvi files on screen. Its use can result in dumping core. If you want to try to use it on this manual, from the system prompt type:

dviiris manual.dvi

Left mouse button---use this to drag the page, and thus adjust the vertical and horizontal position. Place the mouse over the spot you want to move, press the left mouse button, and move the mouse to where you want that spot to be. Release the mouse button, and the page should be redrawn.

Middle mouse button---Used to go to the next position on the page. See the <CR> command.

<SPACE>---used to go to the next page.

<BS>, <DEL>---used to go to the previous page.

<CR>---used to go to the next position on the page. The positions are stored in registers numbered 1-9 as three groups of three (1,2,3), (4,5,6), and (7,8,9). If you are in position 1, hitting <CR> or the middle mouse button will move you to position 2. If you are in position 3, you will be moved back to position 1.

1, 2, 3, 4, 5, 6, 7, 8, 9---go to the position stored in the corresponding register.

S---save the current position in a register specified by the digit key you next press. (Thus, to save the current position as position 5, type S5.) This does not change your position, nor does it change your position group; if you were in position 1, you remain in position 1.

P---go to a particular page. You will be prompted for the page number. If a second number follows the page number, it will be interpreted to mean that there are more than one page with the page number, and to use this one; for instance, 4 2 means to find the second occurance of page 4.

F---open and display a new file (or the first file, if no file name was given on the command line.) You will be prompted for the file name.

R---reopen the same file and go to the current position in it. Used if the current dvi file was updated for some reason.

I---increase magnification one half magstep.

D---decrease magnification one half magstep.

C---clear the screen.

B---toggle the border on/off (default: off).

!---used to exit to a shell. Can be used to re-run TeX, for instance.

Q, X---exit. You must hit one of these two keys twice to exit.

?, H---help. Print this help message.

```
        dviiris specials

        The special commands recognized by dviiris consist of:

landscape---indicates that this document is a landscape document, and the
        box should be printed accordingly.


foreground (color)---indicates the color to use for text and rules.  (color)
        is replaced by the name of a color in lower case, no parenthesis.

background (color)---indicates the color to use for the background.

map (color) (red) (green) (blue)---creates a new color map entry for a
        color you specify.  If the color already exists, this command will
        be ignored.  (red), (green), and (blue) are integers between 0 and
        255 for the red, green and blue intensity levels.
Hit [RETURN]---
```